Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tomaž Žniderič

# Povezovanje androidne aplikacije s spletnim ogrodjem Django

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: viš. pred. dr. Marko Privošnik

Ljubljana 2015

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Sodobne mobilne in spletne aplikacije pogosto shranjujejo svoje podatke v oblaku. Zasnujte in izvedite sistem, ki uspešno povezuje tovrstni aplikaciji. Preučite obstoječe tehnologije, ki so potrebne za izvedbo takega sistema. Pri zasnovi sistema izberite najbolj primerne tehnologije in pristope. Izdelajte ustrezni aplikaciji in preizkusite ter opišite njuno delovanje in uporabo.

# Izjava o avtorstvu diplomskega dela

Spodaj podpisani Tomaž Žniderič sem avtor diplomskega dela z naslovom:

*Povezovanje androidne aplikacije s spletnim ogrodjem Django*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Marko Privošnik,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 8. september 2015 Podpis avtorja:

*Ob tej priložnosti se zahvaljujem mentorju dr. Marku Privošniku za čas in strokovno pomoč pri pisanju diplomske naloge.*

*Zahvaljujem se sestri Tini za pomoč, bratu Mateju za nasvete in sestri Sari za spodbude pri izdelavi diplomske naloge.*

*Posebna zahvala gre staršema za podporo med študijem.*

# Kazalo

# Seznam uporabljenih kratic

| kratica | angleško | slovensko |
|---|---|---|
| **API** | Application Programming Interface | Vmesnik za programiranje aplikacij |
| **HTTP** | HyperText Transfer Protocol Language | Protokol za prenos besedil |
| **ID** | Identification | Identifikacija |
| **JSON** | JavaScript Object Notation | JavaScript zapis objektov |
| **ORM** | Object-realation mapping | Objektno-relacijsko mapiranje |
| **POS** | Point-of-sale | POS sistem |
| **REST** | Representational State Transfe | Predstavitveni prenos stanja |
| **SDK** | Software development kit | Programski razvojni paket |
| **SOA** | Service-oriented architecture | Storitveno orientirana arhitektura |
| **SOAP** | Simple Object Access Protocol | Protokol za preprost dostop objektov |
| **SQL** | Structured Query Language | Strukturiran jezik poizvedb |
| **SSL** | Secure Sockets Layer e | Plast varnega vtiča |
| **URL** | Uniform resource locator | Enolični krajevnik vira |
| **XML** | Extensible Markup Language | Razširljiv označevalni jezik |

# Povzetek

Cilj diplomskega dela je predstavitev razvoja med seboj povezane spletne in mobilne aplikacije ter predstavitev tehnologij za njuno medsebojno povezovanje. Pri tem gre za sistem, v katerem so podatki shranjeni na strežniku, do njih pa je mogoče dostopati preko več različnih naprav. Prvi del diplomskega dela je namenjen predstavitvi orodij in tehnologij, ki so bila uporabljena pri implementaciji Android in spletne aplikacije. Poleg tega so v tem delu predstavljena tudi druga orodja in pristopi za implementacijo takšnega sistema, prikazana pa je tudi njihova primerjava.

Drugi del diplomskega dela opisuje praktično izvedbo takšnega sistema. Ta del opisuje arhitekturo aplikacije ter funkcionalnosti posameznih delov. Poleg tega opisuje tudi pristope in uporabe tehnologij za uspešno povezovanje mobilne aplikacije s spletno. Predstavljena je tudi ideja za ažuriranje podatkov ter način avtentikacije v sistem.

**Ključne besede:** Django, Android, spletna ogrodja, mobilne aplikacije, spletne storitve, REST.

# Abstract

The main goal of the following thesis is the presentation of development of the web and mobile applications and demonstration of technologies for their successful interconnection. Such system stores all data on server and may be accessed through many different devices. The first part of the thesis is devoted to the presentation of tools and technologies that have been used for the successful implementation of such system. The theoretical part of the thesis also presents other tools and technologies for implementation of such system and presentes their comparison.

The second part of the thesis contains practical implementation of such system. This part describes the application architecture and functionality of each part. This part of the thesis also describes approaches and used technologies for the successful integration of mobile applications and web application. In addition, this part describes the idea of syncing data on mobile application, and desribes the methods to authentication.

**Keywords:** Django, Android, Web framework, mobile application, web services, REST.

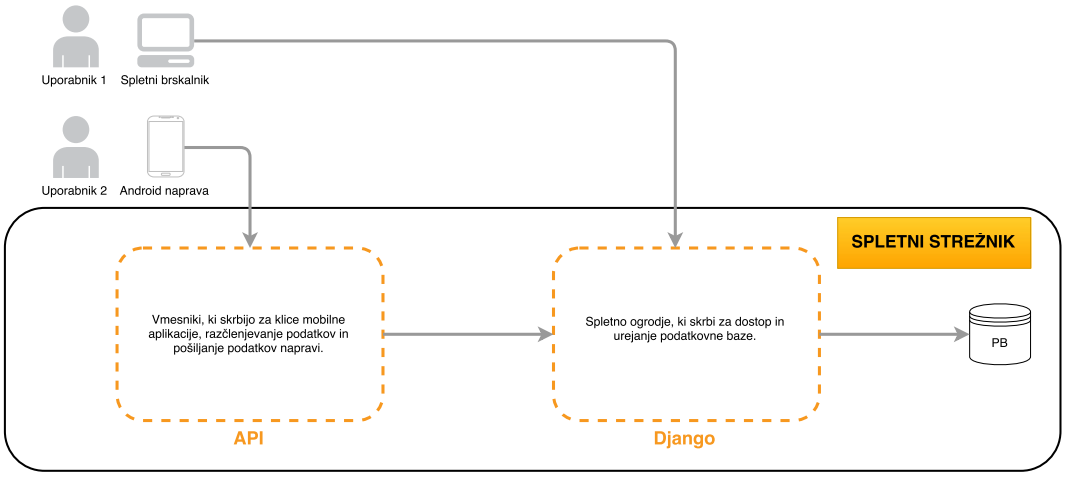
# Poglavje 1

# Uvod

V današnjih časih je porast spletnih in mobilnih aplikacij vse večji. Njihova funkcionalnost lajša uporabnikova vsakodnevna opravila. Vse več uporabnikov pa želi dostopati do istih podatkov preko različnih naprav, kot sta mobilni telefon in računalnik. Podatki uporabnikov so običajno shranjeni v oblaku na strežniku, dostop do teh podatkov pa mora zagotoviti razvijalec programske opreme.

Tu se pojavi dilema pri izbiri ogrodja za lažje razvijanje spletnih aplikacij, tehnologiji za povezovanje mobilnih aplikacij ter izbiri pristopa pri razvoju mobilnih aplikacij. Diplomsko delo opisuje različne pristope in tehnologije za povezovanje spletne aplikacije z androidno. V njem so predstavljena ogrodja za hitrejše razvijanje spletnih aplikacij ter njihova primerjava. Poleg tega opisuje in primerja različne tehnologije za implementacijo spletnih storitev. Utemeljuje pa tudi izbiro mobilne platforme za razvoj in način izdelovanja.

V diplomskem delu je v osredje postavljeno ogrodje Django, ki služi kot ogrodje za razvoj spletnih aplikacij, in mobilna platforma Android, v katerem je razvit praktičen primer mobilne aplikacije. Za implementacijo takšnega sistema je potrebno razviti posebne spletne storitve, ki jim rečemo APIji. Ti so razviti znotraj spletne aplikacije ter skrbijo, da lahko do istih podatkov dostopamo preko različnih platform. S takšnim sistemom lahko uporabniki dostopajo do svojih podatkov tako iz Android naprave kot tudi brskalnika.




Slika 1.1: Dostopa do spletne aplikacije preko različnih naprav.

V drugem poglavju najdemo primerjavo izrazov kot so spletna stran, spletna aplikacija, mobilna aplikacija ter spletne storitve.

V tretjem poglavju diplomskega dela se srečamo z vso izbrano tehnologijo, protokoli in uporabljenimi orodji za uspešen razvoj mobilne ter spletne aplikacije. Poleg tega so predstavljene tehnologije za izmenjavo podatkov in povezovanje med spletno in mobilno aplikacijo. Poglavje primerjava tehnologij podrobno predstavi različne pristope, ogrodja in tehnologije za razvoj aplikacij. Ta del tudi utemeljuje odločitve, ki sem jih sprejel za uspešno implementacijo aplikacij.

Peto poglavje na kratko predstavi aplikacijo, ki je uporabljena za primer implementacije povezovanja Android aplikacije s spletnim ogrodjem Django. Tu je na kratko opisana ideja aplikacije in primerjava podobnih alternativnih aplikacij, ki so že na trgu.

V šestem poglavju je predstavljena arhitektura aplikacije. Notri lahko najdemo kako je aplikacija razdeljena ter kakšne funkcionalnosti ti deli predstavljajo.

V poglavju Razvoj aplikacije najdemo štiri obširnejše pristope k implementaciji aplikacije. V prvem pristopu je predstavljena implementacija pove-

zovanja mobilne aplikacije s spletno, v drugem je predstavljena avtentikacija uporabnika, v tretjem ažuriranje podatkov med spletno in mobilno aplikacijo, v zadnjem pa je predstavljen pristop generiranja ukazov za tiskanje na termični račun.

# Poglavje 2

# Spletne storitve in odjemalci

Spletne storitve so deli spletne aplikacije. So standardiziran način integriranja spletnih aplikacij za izmenjavo podatkov med elektronskimi napravami preko internetnih protokolov. Še nekaj let nazaj, spletne storitve niso bile dovolj hitre, da bi bile zanimive, dandanes pa se v praksi vse bolj uporabljajo [12].

Ko govorimo o spletnih storitvah, pojma ne smemo zamešati s spletnimi aplikacijami ali spletnimi stranmi, se pa ti pojmi prepletajo med seboj. Ko govorimo o spletnih straneh, govorimo o statični predstavitvi podatkov. Do spletnih strani uporabnik dostopa preko spletnih brskalnikov, ker je vsebina statična, pa nima interakcije s podatki na njej. Uporabniku je predstavljena vsebina, ki je navadno enaka za vse.

Če želimo našo spletno stran nadgraditi tako, da uporabniku omogoča večjo funkcionalnost, nadzor in interakcijo, moramo uporabiti veščine programiranja. Ko naši spletni strani dodamo funkcionalnosti, kot je na primer prijava uporabnika v sistem, se naša spletna stran spremeni v spletno aplikacijo. Vsebina podatkov v spletni aplikaciji ni več statična, saj se od vsakega uporabnika razlikuje. Uporabnik ima v spletni aplikaciji ločene podatke od ostalih uporabnikov, ti podatki pa so shranjeni na strežniku. Primer spletne aplikacije je koledar, v katerega si uporabnik beleži svoje podatke ter do njih dostopa, medtem ko so na spletni strani, kjer se na primer objavljajo

novice, podatki statični in enaki za vse uporabnike. Ko uporabljamo pojem spletna aplikacija, še vedno govorimo o dostopu preko spletnih brskalnikih tako kot pri spletnih straneh. Spletna aplikacija poleg podatkov uporabniku vrne tudi grafični izgled ter celotno postavitev spletne strani. Za podatke in uporabniški vmesnik pri spletnih aplikacijah skrbi strežnik. Pojma spletne aplikacije pa ne smemo mešati z Android aplikacijami, javanskimi aplikacijami ali drugimi, ki za prikaz podatkov uporabljajo spletne storitve, te pa so običajno del spletnih aplikacij.

Spletne storitve se od spletnih aplikacij razlikujejo v tem, da spletne storitve skrbijo za prenos dinamičnih podatkov, medtem ko spletna aplikacija poleg tega skrbi še za grafični vmesnik. Tako lahko rečemo, da so spletne storitve del spletnih aplikacij, ki omogočajo dostop do strežnika in podatkov preko različnih internetnih protokolov. Za izdelavo spletne aplikacije implementacija spletnih storitev ni potrebna, ker pa je danes velika porast mobilnih naprav, pa je ta vse bolj zaželena. Če želimo, da naša mobilna aplikacija dostopa do istih storitev kot naša spletna, je potrebno za realizacijo takega sistema vzpostaviti storitveno orientirano arhitekturo (angl. SOA - service orientated architecture). Naši spletni aplikaciji implementiramo različne vmesnike uporabniškega programa (angl. API - Application Programming Interface), ki skrbijo za povezovanje drugih aplikacij s spletnim strežnikom. Spletne storitve skrbijo za branje podatkov, ki jih aplikacija pošlje, obdelavo teh podatkov ter vračanje podatkov naši aplikaciji. Spletne storitve skrbijo le za dinamične podatke ter aplikacijam ne vračajo grafične podobe, saj mora to v celoti izdelati programer.

Do spletnih storitev lahko dostopa tudi spletna aplikacija, vendar mora biti za prikaz podatkov že v naprej poskrbljeno. Primer dostopa spletne aplikacije do spletnih storitev je, ko je zaradi velikega števila podatkov uporabljeno številčenje strani. Če uporabnik želi videti naslednje podatke, naša spletna aplikacija kliče spletno storitev, ki poskrbi samo za vsebino teh podatkov in ne za celotni prikaz spletne strani. Tako kot za implementacijo spletne aplikacije ni potrebna implementacija spletnih storitev, tudi za im-

plementacijo spletnih storitev ni potrebna implementacija spletne aplikacije. Ker je dandanes razširjenost spletnih brskalnikov ter mobilnih naprav velika, pa je ta zaželena. Za implementacijo spletnih storitev je bilo poskrbljeno že v preteklosti, saj se je že leta 1998 pojavil protokol SOAP[11]. Ker pa se je ta protokol pojavil pred velikim porastom mobilnih aplikacij, pa se je danes kot odgovor pojavil predstavitveni prenos podatkov REST, ki za razliko od SOAP ni protokol, uporablja pa druge internetne protokole za prenos podatkov. Oba pa imata isto lastnost prenosa strukturiranih podatkov.

Spletni odjemalci so aplikacije, ki dostopajo do spletnih storitev oziroma do spletne aplikacije. Mednje štejemo spletni brskalnik, Android aplikacije, iOS aplikacije, različne javanske aplikacije in druge. Skupna lastnost odjemalcev je, da se za prenos podatkov povezujejo s strežnikom preko različnih protokolov. Prednost spletnih storitev je, da se lahko odjemalci povezujejo do istih APIjev, s tem pa se izognemo večkratnemu programiranju iste funkcionalnosti. V večini primerov za vizualizacijo podatkov v aplikaciji poskrbi programer. Aplikacija se poveže s spletnim strežnikom, od koder pridobi samo vsebinske podatke, nato pa sama poskrbi za izgled aplikacije. Temu pa ni vedno tako, saj na primer spletni brskalniki ali hibridna aplikacija Android za prikaz podatkov pridobita s strani strežnika še potrebne HTML in CSS datoteke, ki skrbijo za prikaz teh podatkov.

# Poglavje 3

# Uporabljena orodja in tehnologije in protokoli pri izdelavi aplikacije

V tem poglavju si bomo podrobneje pogledali vse tehnologije in programe, ki sem jih uporabil pri izdelavi aplikacije. V prvem delu bom predstavil vse tehnologije, ki so bile uporabljene za izdelavo same Android aplikacije. Predstavil bom operacijski sistem Android ter v tem okviru tudi razvojno orodje Eclipse, v katerem je bila aplikacija napisana. Poleg tega bom predstavil še format zapisa za izmenjavo podatkov med strežnikom in klientom JSON ter podatkovno bazo SQLite. V drugem delu pa bom predstavil spletno ogrodje Django, v katerem je bila napisana spletna aplikacija.

Ker fakulteta za računalništvo in informatiko veliko časa nameni programskemu jeziku Javi in Python, mi jezika nista bila tuja. Prav tako se je na študiju možno seznaniti z izdelavo mobilnih aplikacij, kar mi je dajalo odlično izhodišče za razvoj tovrstne aplikacije. Nova izkušnja pa je bila ogrodje Django, katero zaradi svoje funkcionalnosti postaja vse bolj razširjen način izdelave spletnih aplikacij.

Za razvoj Android aplikacije sem uporabil razvojno orodje Eclipse, ki je bilo do sedaj priporočeno orodje za razvoj tovrstnih aplikacij. Mobilno

aplikacijo se razvija v programskem jeziku Java, ki je eden izmed vodilnih jezikov danes. Za razvoj aplikacije je bilo potrebno namestiti tudi programski razvojni paket Android SDK, za integracijo tega paketa in Eclipsa pa je skrbel vtičnik ADT. Ker je podjetje želelo, da aplikacija deluje čim hitreje, sem se moral v tem okviru seznaniti s podatkovno bazo SQLite, ki je skrbela, da so bili podatki shranjeni tudi lokalno in jih ni bilo potrebno vnovično prenašati. Poleg te podatkovne tehnologije je za razvoj Android aplikacije potrebno poznavanje XML, saj ta skrbi za izgled in konfiguracijo aplikacije.

## 3.1 Android

Android je mobilni operacijski sistem, ki temelji na Linuxovem jedru [2]. Uporablja se predvsem na napravah z zaslonom za dotik, kot so mobilni telefoni in tablični računalniki. Ukazi, ki se odzivajo na dotik, kot so pritiski, potegi in podobno zagotavljajo odlično uporabniško izkušnjo. Zaradi vsestranske uporabnosti se je Android razširil tudi na televizije, avtomobile ter ročne ure. Vse bolj pa se uporablja tudi na osebnih računalnikih in igralnih konzolah.

Android je odprtokodni operacijski sistem. S tem je tudi brezplačen za uporabo in razvoj aplikacij. Zaradi tega in same razširjenosti operacijskega sistema je popularen predvsem pri podjetjih, ki bi radi razvili aplikacijo za zmogljive naprave s prilagodljivim operacijskim sistemom ter si za razvoj aplikacije ne bi po nepotrebnem višali stroške. Po raziskavah je Android vodilna platforma za razvoj aplikacij in ima največji delež razvijalcev. Prav tako večina razvijalcev vidi Android kot primarno platformo za razvoj [20].

### 3.1.1 Prednosti Androida

Operacijski sistem Android je trenutno najbolj razširjen operacijski sistem na trgu. K temu zagotovo pripomore njegova odprtokodnost, kar je podjetju Blocklogic d.o.o predstavljalo cenejše in enostavnejše razvijanje programov. Ker je operacijski sistem najbolj razširjen tako na svetovnem trgu kot tudi

v Sloveniji, je podjetju predstavljalo tudi večji tržni delež uporabnikov. Ker so ciljna publika podjetja tisti, ki bi za uporabo storitev radi varčevali, je ta rešitev tudi bolj ugodna, saj so naprave, ki poganjajo sistem Android, običajno cenejše in tako končnim uporabnikom predstavljajo manjši strošek.

Prednost Androida predstavlja tudi razširjena trgovina Google Play, ki omogoča uporabnikom enostavno nalaganje in posodabljanje svojih aplikacij. Uporabnik lahko preko ključnih besed najde aplikacije v tej trgovini in jo nato prenese z Googlovega strežnika. Namestitev in posodobitev aplikacij pa je avtomatiziran proces in končnim uporabnikom omogoča, da imajo na svoji napravi nameščeno tekočo različico aplikacije.

## 3.2 SQLite

SQLite je podatkovna baza, ki ne potrebuje ločenega strežnika za svoje delovanje. Je sestavni del aplikacije, za svoje delovanje pa uporablja lokalni prostor, kamor je shranjena kot datoteka. Ima običajne lastnosti ostalih SQL podatkovnih baz. Tako vsebuje različne tabele, poglede, SQL sintakso transakcij in drugo. SQLite je priljubljen, saj je brezplačen za kakršnokoli uporabo, tako komercialno kot zasebno [3]. Prav tako podatkovna baza porabi malo pomnilnika, podprta pa je za različne programske jezike. Njena prednost je tudi v tem, da za svoje delovanje ne potrebuje nobene dodatne namestitve in konfiguracije. Za svoje delovanje potrebuje le datoteko do katere dostopa, zato je ta podatkovna baza tudi zelo prenosna. Odlično se odreže takrat, ko do nje ne dostopa več uporabnikov hkrati, saj se datoteka v kateri je shranjena podatkovna baza za vsako transakcijo zaklene.

Zaradi teh lastnosti je podatkovna baza SQLite primerna za razvijalce Android. Ker je uporabnik naprave ena sama oseba, ki ima svojo podatkovno bazo, se ta odlično odreže tako v hitrosti kot tudi uporabnosti. Operacijski sistem Android ima vključene APIje za dostop do SQLite podatkovne baze, ki razvijalcu lajšajo delo s podatkovnimi bazami.

## 3.3 XML

Kratica XML je okrajšava za angleški izraz Extensible Markup Language [4]. Je preprost in fleksibilen format teksta. Po sintaksi je podoben jeziku HTML. Velikokrat se uporablja za prenašanje podatkov med strežnikom in spletom. XML je razširljiv označevalni jezik, saj ima možnost, da si lahko sami izmislimo imena etiket (angleško TAG).

```
<Knjige>
    <Knjiga>
        <Naslov>Naslov knjige</Naslov>
    </Knjiga>
</Knjige>
```

Slika 3.1: Primer xml datoteke.

V androidu se jezik uporablja predvsem pri izdelovanju izgleda aplikacije. Sem spada postavitev gradientov, ki jih aplikacija vsebuje, ter pisanju stilov (spreminjanje barv, razne grafične podobe, izbiranje pisav, animacije), ki sami aplikaciji dajejo končni izgled. Uporablja se tudi za osnovno konfiguriranje aplikacije.

## 3.4 JSON

JSON je format zapisa za izmenjavo podatkov. Je preprosta oblika besedila, ki pa je bila zasnovana tako, da je neodvisna od programskih jezikov [15]. Tako jo podpirajo vsi moderni jeziki, kot so C, C++, Java, JavaScript, Python ter drugi. Zaradi teh lastnosti je JSON univerzalna oblika zapisa za izmenjavo podatkov.

V JSON zapisu je mogoče zaslediti dve strukturi:

- Ključ/vrednost. Ta struktura ima vsakemu ključu prirejeno eno vrednost in tako zelo spominja na razpršeno tabelo (Hash map).

- Seznam vrednosti. V strukturi je množica podatkov, do katerih lahko dostopamo preko kazalca. Struktura je podobna navadnemu seznamu.

Seveda pa JSON dopušča možnost kombinacije teh dveh podatkov.


Slika 3.2: Primer JSON oblike zapisa.

JSON je tehnologija, ki sem jo uporabil za izmenjavo podatkov med strežnikom in mobilno napravo. Ker imata Python in Java že vključene knjižnice za upravljanje s tem podatkovnim tipom zapisa, mi ni bilo potrebno dodatno prenesti ali konfigurirati ničesar. Knjižnici sta tudi zelo razširjeni, zato je branje in kreiranja zapisa JSON enostavno.

## 3.5 Java

Java je programski jezik, ki so ga v podjetju Sun Microsystems začeli razvijati leta 1990. Različica Java 1.0 je bila objavljena leta 1996 in se še do današnjega dneva redno posodablja. Je preprost, objektivno orientiran, robusten, varen in visoko zmogljiv programski jezik. Java je neodvisen od

sistema, saj se aplikacije poganjajo z navideznim strojem (JVM). Zaradi tega lahko aplikacija teče na več platformah [13].

Java je priporočen programski jezik pri pisanju Android aplikacij, saj teče na navideznem stroju in je ni potrebno vsakič znova prevesti za vsako napravo posebej. Android je zasnovan v programskem jeziku Java, s tem pa ima razvijalec dostop do Android knjižic preko orodja Android SDK. Ta lastnost povečuje tudi varnost aplikacij. Prav tako je Java zelo razširjen jezik, ki vsebuje veliko svojih knjižic za bolj učinkovito uporabo.

## 3.6 Eclipse

Eclipse je integrirano razvojno okolje (IDE), ki temelji na Javi. Je odprtokoden in brezplačen. Sestavljen je iz razvojnega okolja ter številnih naborov razširitev, ki razvijalcu omogočajo hitrejše razvijanje programske kode. Zaradi tega sistema naborov razširitev, je poleg Jave v njem mogoče pisanje programske kode tudi v drugih jezikih kot so: C, C++, COBOL, Javascript, PHP, itd. [14]. Pred začetkom razvijanja aplikacije, pa je za delovanje potrebno prenesti razvojni paket Android SDK ter vtičnik ADT za integracijo med Eclipsom in tem paketom.

Eclipse je bil do nedolgo nazaj priporočeno okolje za razvoj mobilnih aplikacij. To okolje je sedaj zamenjalo Android Studio. Za okolje Eclipse sem se odločil, ker je bila ob začetku razvoja mobilne aplikacije izdana le testna različica Android Studia.

### 3.6.1 Android SDK

Programski razvojni paket Android SDK je skupek razvojnih orodij, ki omogočajo razvijanje aplikacij. Med ta orodja spadajo razhroščevalnik, knjižnice, navidezni stroj, dokumentacija, primeri kod in vodiči [16].

Do ne dolgo nazaj je bilo uradno podprto integrirano razvojno okolje Eclipse, ki za delovanje potrebuje še vtičnik razvojnih orodij ADT. Od leta

2015, pa je za razvoj aplikacij uradno podprt Android Studio, ki ga je izdal Google.

### 3.6.2 Vtičnik ADT

Android development Tools je vtičnik, ki ga je razvil Google z namenom, da Eclipsu zagotavlja integrirano okolje v katerem se razvija Android aplikacije. Ta razširitev omogoča Eclipsu ustvarjanje novih projektov ter ponuja preproste poglede uporabniških vmesnikov. Preko njega je mogoče dodajati nove pakete, ki temeljijo na Androidovih API-jih, razhroščevanje aplikacije ter izvoz .apk datotek.

## 3.7 Python

Python je visokonivojski programski jezik, ki je zelo razširjen. Programski jezik je značilen po svoji enostavnosti, kar omogoča razvijalcem hitrejše učenje jezika, ter pisanju programov v manj vrsticah kot v drugih programskih jezikih kot je na primer Java. Programski jezik je objektivno orientiran, njegovi podatkovni tipi pa so popolnoma dinamični. Velikokrat se jezik Python uporablja za pisanje spletnih aplikacij.

Strežniški del spletne blagajne je napisan v Pythonu. Za povezovanje mobilne aplikacije s spletno je potrebno napisati napisati APIje, do katerih je aplikacija dostopala. Za razvoj spletne aplikacije pa skrbi ogrodje Django.

## 3.8 Django

Django je brezplačno odprtokodno ogrodje za izdelavo spletnih aplikacij. Napisan je v programskem jeziku Python ter temelji na principu model-pogled-nadzornik (angl. model-view-controller). Glavni namen ogrodja je olajšanje in pohitritev izdelave spletnih aplikacij. Omogoča tudi preprosto povezavo spletne aplikacije s podatkovno bazo.

## 3.9 PostgreSQL

PostgreSQL je zmogljiv sistem za upravljanje s podatkovnimi bazami. Je objektivno relacijski sistem, ki ponuja tradicionalne lastnosti tovrstnih sistemov [18]. Svojo moč mu daje več kot 20 let aktivnega razvoja, je tudi popolnoma konkurenčen ostalim bolj znanim izdelkom (npr. DB2, MSSQL ali Oracle), vendar se od njih razlikuje v tem, da je PostgreSQL odprtokoden in tako tudi brezplačen.

PostgreSQL je Spletni blagajni služil kot primarni sistem za dostop do podatkovne baze na strežniškem delu. Zaradi svoje zmogljivosti je idealna rešitev, saj je v spletni aplikaciji veliko tabel in vnosov, ki potrebujejo hitro, stabilno in zanesljivo delovanje.

## 3.10 PyCharm

Pycharm je integrirano razvojno okolje (IDE) in je namenjen programiranju v jeziku Python. Na prvi pogled zelo spominja na Eclipse, ponuja pa analiziranje kode, grafični razhroščevalnik, testiranje modulov ter omogoča popolno podporo za ogrodje Django. Zaradi te podpore smo ga pri podjetju določili kot ogrodje za razvoj spletne aplikacije.

## 3.11 Django REST framework

Django REST framework je močno in fleksibilno ogrodje, ki nam olajša delo z APIji. Ima vgrajeno podporo za razločne avtentikacije, dovoljenja, kontrole različic in druge politike za ApIje. Je preprost za uporabo ter olajša delo razvijalcem. Django REST framework je odprtokoden.

V spletni aplikaciji se je Django REST framework uporabljal za avtentikacijo z žetonom, pisanjem APIjev za dostop mobilne aplikacije ter upravljanjem s podatkovnim tipom JSON.

# Poglavje 4

# Primerjava tehnologij

Za izdelavo mobilne in spletne aplikacije je možno uporabiti veliko različnih tehnologij in orodij. Izbiramo lahko med načini povezovanja mobilne aplikacije s spletno, uporabo različnih orodij za hitrejše izdelavo aplikacij ter uporabo različnih orodij in programskih jezikov za izdelavo spletne aplikacije. V tem delu diplomskega dela bom primerjal različne tehnologije in utemeljil izbire tehnologij, ki sem jih uporabil.

## 4.1 Primerjava spletnih ogrodij

Popularnost spletnih aplikacij danes raste. V te namene se tudi razvijajo ogrodja, ki omogočajo lažje in hitrejše pisanje teh aplikacij. Med najbolj popularne spadajo Ruby on Rails, Django, Laravel ter drugi. Ogrodja je težko neposredno primerjati med seboj, vsa našteta pa so s strani programerjev zelo priljubljena. Prav tako je zelo težko reči, da je eno ogrodje boljše od drugega. Razlikujejo se v programskih jezikih, sintaksi programiranja, hitrosti delovanja in drugem. Skupna lastnost teh ogrodij je, da so MVC ogrodja. To ime in lastnosti lahko razdelimo na tri dele[9]:

- model, ki predstavlja podatke v povezavi s podatkovno bazo. Ta skrbi za branje in shranjevanje podatkov.
- pogled, ki prikazuje podatke iz modela

- kontroler, ki skrbi za interakcijo z uporabnikom. Sem spada branje podatkov iz pogleda, ter pošiljanje podatkov modelu.

### 4.1.1 Programski jezik

Prva velika razlika med ogrodji je programski jezik. Ko se razvijalec odloča za izbiro ogrodja, se mora v prvem koraku naučiti jezik, v katerem bo razvijal. Jeziki se razlikujejo po težavnosti učenja, hitrosti izvajanja kode ter po naboru knjižic, ki jih vsebujejo. Prej omenjena ogrodja vsebujejo naslednje programske jezike.

| **Ogrodje** | **Programski jezik** |
|---|---|
| Django | Python |
| Ruby on Rails | Rails |
| Laravel | PHP |

Tabela 4.1: Primerjava programskih jezikov posameznih ogrodij.

### 4.1.2 Težavnost učenja

Na splošno sta PHP in Python preprostejša in hitreje učljiva jezika od Rails [7]. Čeprav je dokumentacija pri vseh treh ogrodjih zelo obširna, Django in Laravel spadata pod lažje učljiva, saj je jezik v katerem se razvija bolj preprost. Djangova dokumentacija pa je najbolj razumljiva in najpreprostejša. Po drugi strani pa lahko veščine Laravel z predznanjem PHPja osvojimo v nekaj tednih.

### 4.1.3 Zmogljivost

Za varnost je pri vseh ogrodjih zelo dobro poskrbljeno. Vendar zaradi površnosti programerji velikokrat delajo napake. Django izmed teh treh velja za najbolj varnega, saj je dokumentacija obširna, sintaksa preprostejša, zato razvija-

lec naredi manj napak [7]. V končnem pa vseeno težko definiramo, katero ogrodje zagotavlja najboljšo varnost, saj mora za to poskrbeti razvijalec sam.

Hitrost izvajanja kode je odvisna od programskega jezika. Tako je v Pythonu napisan Django najhitrejši, medtem ko je Laravel s programskim jezikom PHP najpočasnejši.

Ogrodja se razlikujejo tudi po hitrosti razvijanja aplikacij. Pri preprostejših aplikacijah se najbolje odvije Ruby on Rails, saj ima vgrajen velik nabor bližnjic za razvijanje. Vendar se z večjo kompleksnostjo aplikacij Django bolje odreže, saj je sintaksa Python bolj pregledna in lažje razumljiva, zato razvijalec dela manj napak. Najslabše se odreže Laravel, saj je za implementacijo spletne aplikacije potrebno napisati veliko koda, ta pa je velikokrat nepregledna.

### 4.1.4 Prednosti in slabosti ogrodij

#### Django

Prva glavna prednost Djanga je jezik Python, ki je preprost za učenje in ima zelo pregledno sintakso [8]. Django je bil razvit leta 2006 in je do danes prejel veliko nadgradenj in razširitev, ki omogočajo hitrejši razvoj spletnih aplikacij. Poleg tega ima zelo dobro podprt ORM. Ta skrbi, da se Python razredi razširijo z Djangovimi. Vsak atribut razreda pa predstavlja vrstico v podatkovni bazi. Tako je definiranje podatkovnih baz zelo preprosto. Poleg tega ima Django še prilagodljivo skrbniško ploščo ter vgrajeno avtentikacijo. Dokumentacija Djanga je zelo obširna. Največja slabost, ki jo Django prinaša pa je, da tudi če gre za manjšo aplikacijo, ta zgleda napihnjena in obširna.

#### Ruby on Rails

Največja prednost, ki jo Ruby on Rails prinaša je številčnost razširitev, ki so razviti od tretjih razvijalcev [8]. Te so ponavadi dobro dokumentirane in preproste za uporabo. Poleg tega je Ruby zelo berljiv jezik, vendar pa je težje učljiv. Poleg tega ima Ruby on Rails zelo obširno dokumentacijo in

veliko vodičev. Prednost predstavlja tudi to, da se manjši projekti v tem ogrodju lažje in hitreje razvijajo, vendar kompleksnost raste eksponentno z obširnostjo projekta.

#### Laravel

Laravel ima tako kod Ruby on Rails in Django zelo obširno dokumentacijo [8]. Njegova prednost je dobra podpora REST APIjev, saj se da hitro pisati spletne storitve. Tako kot Django ponuja ORM, ki pomaga organiziranju podatkovne baze. Pomanjkljivost Laravela je to, da ogrodje ni samostojno, ampak odvisno od drugih ogrodij. To lahko vpliva na Laravel, saj morajo razvijalci Laravel poskrbeti za spremembe, če do njih pride pri drugih ogrodjih.

### 4.1.5 Razširjenost

Čeprav je Ruby on Rails najzahtevnejši, pa je povpraševanje po njihovih programerjih največje. Prav tako so projekti, ki so izpeljani v tem ogrodju bolj razširjeni in bolj poznani. Ogrodje, ki si od teh treh zasluži najmanj omembe pa je Laravel, saj je povpraševanje in število poznanih projektov najmanjše.

| **Ogrodje** | **Spletne Aplikacije** |
|---|---|
| Django | Pinterest, The Guardian, The Onion |
| Laravel | Deltanet Travel |
| Ruby | Github, Airbnb, Ask.fm, Hulu |

Tabela 4.2: Uporaba spletnih ogrodij v produkciji.

Za implementacijo naše spletne aplikacije smo izbrali ogrodje Django, saj je direktor podjetja z njim najbolje seznanjem.

## 4.2 Primerjava med SOAP in REST

Za povezovanje drugih platform s spletnimi storitvami lahko uporabimo različne pristope. Najbolj poznana danes sta SOAP in REST, pojavlja pa se dilema med izbiro enega izmed njih.

SOAP je zelo poznana tehnologija, ki se uporablja znotraj SOA ogrodja [**?**]. Je specifikacija protokola za izmenjavanje strukturiranih podatkov v obliki XML. Njegova glavna prednost je, da omogoča uporabo različnih protokolov za izmenjavo podatkov. Poleg sinhronega HTTP, SOAP podpira tudi asinhrone vrste, uporabimo pa lahko tudi protokol SMTP ali storitev kot je na primer JMS. Protokol SOAP je bil zasnovan leta 1998, kar pa je pred velikim porastom mobilnih aplikacij.

Prednosti SOAP so tudi uporaba WS-Security, ki protokolu omogočajo dodatne funkcionalnosti na področju varnosti [6]. Velikokrat pride do zmote, da je zaradi tega SOAP bolj varen od REST, saj se te dodatne funkcionalnosti le redko uporabljajo. Poleg tega SOAP podpira WS-AtomicTransaction, ki zagotavlja ACID transakcije. Take transakcije zagotavljajo pravilno izvršitev v primeru, da se transakcija ne izvrši v celoti, v primeru izpada elektrike in podobno. SOAP podpira tudi WS-ReliableMessaging, ki skrbi, da se transakcija ponovi, če ta ni bila uspešna. Funkcionalnosti SOAP so zelo uporabne, vendar za dostop do spletnih storitev velikokrat nesmiselne. Tehnologije, ki jih ponuja SOAP so uporabne takrat, ko delamo z bančnimi transakcijami, saj potrebujemo tam večji nadzor.

REST pa predstavlja arhitekturo, ki se uporablja za izmenjavanje strukturiranih podatkov v različnih preprostejših oblikah. Za razliko od SOAP, REST ni protokol, vendar opredeljuje, kako uporabljati obstoječe standarde. Za dostopanje do storitev uporablja protokol HTTP.

Glavna prednost REST je, da uporablja protokol HTTP, ki je bolj preprost za uporabo [6]. Tako je implementacija na strani odjemalca in strežnika bolj preprosta. Prav tako REST za izmenjavo podatkov uporablja različne formate kot so navaden tekst, HTML, XML, JSON in drugi. REST uporablja tudi manjšo pasovno širino ter porabi manj virov za izmenjavo podatkov. Če

želimo implementirati SOAP v Androidu, moramo uporabiti knjižice drugih razvijalcev, medtem ko pri REST uporabimo že integrirane javanske knjižice.

Glavne razlike REST in SOAP so predstavljene v tabeli [6].

| **SOAP** | **REST** |
|---|---|
| SOAP je protokol | REST je arhitekturni stil |
| SOAP je zahtevnejši za uporabo in težji za implementacijo | REST je preprostejši in lažji za implementacijo, prav tako ima Java že vgrajene knjižice za uporabo |
| SOAP uporablja standarde, ki jim moramo točno slediti | REST ne uporablja prevelikega nabora standardov, je preprostejši |
| SOAP uporablja večjo pasovno širino in več virov | REST uporablja manjšo pasovno širino in manj virov |
| SOAP ima zagotovljeno svojo varnost | REST uporablja varnost drugih protokolov |
| SOAP dovoljuje samo XML format zapisa | REST dovoli različne zapise formata kot so tekst, HTML, XML, JSON in drugi |
| SOAP je manj priporočen | REST je vse bolj priporočen |

Tabela 4.3: Primerjava tehnologij SOAP in REST.

Ker so za uporabo REST v Androidu že vgrajene knjižice, je hitrejši za obdelavo in prenos podatkov, ponuja več vrst serializacij podatkov ter je priporočen z več strani razvijalcev, sem se odločil za slednjega.

Za implementacijo REST spletnih storitev v Djangu lahko uporabimo več ogrodij. Dva izmed bolj popularnih sta Django REST Framework ter Tastypie. Oba ogrodja omogočata razvijalcu preprosto implementacijo različnih APIjev. Razlikujeta se v POS strukturah ter načinu uporabi, vendar sta obe ogrodij zelo močni. Glavne prednosti Django-REST Framework je podpora oAuth2, ki skrbi za avtentikacijo, boljša dokumentacija, ogrodje je lažje za uporabo, za integracijo pa ni potrebno veliko dodatnega dela.

## 4.3 Izdelava mobilne aplikacije

Za izdelavo mobilne aplikacije si moramo v prvem koraku izbrati platformo. V podjetju smo se odločili za platformo Android, saj smo hoteli, da je naša aplikacija pokriva čim večji delež uporabnikov, ter da naprave katere bi uporabljali končni potrošniki ne predstavljajo prevelikega stroška.

Pri izdelavi Android aplikacije lahko izbiramo med dvema načinoma. Razvijamo lahko izvirne (angl. native) ali hibridne aplikacije. Pri prvi ima razvijalec popoln nadzor nad razvijanjem aplikacije, medtem ko se mora pri drugi zanašati nad HTML tehnologijo. Razvijanje hibridnih aplikacij je hitrejše, k čemur zagotovo pripomore tudi ogrodje Django okolje, saj ima ta možnost vračanja HTML datotek in njihovih stilov za izgled aplikacij. Ker pa smo v podjetju hoteli popoln nadzor in celotno funkcionalnost pri razvoju smo se odločili za izvirno implementacijo.

Za razvoj hibridnih aplikacij lahko uporabimo pospeševalnike za razvoj aplikacij, med katere štejemo: PhoneGap, Ionic, Mobile Angular UI, Appcelerator ter druge. Glavna prednost teh pospeševalnikov je hitrejši razvoj aplikacij, kompatibilnost takih aplikacij na več platformah (iOS, Android) ter s tem krajšanje časa in stroškov razvoja aplikacij. Skupna lastnost takih pospeševalnikov je, da za razvoj aplikacij uporabljamo jezik HTML. Ker pa smo pri teh pospeševalnikih omejeni zgolj na tehnologijo HTML, pa ne moremo uporabiti različnih standardov, ki so podprti v izvirni implementaciji aplikacij. Zaradi tega je včasih težko izdelati lep vmesnik, izkušnja uporabnikov pa je omejena na tehnologijo HTML. Hibridne aplikacije so uporabne zlasti takrat, ko projekt ni preobširen, vmesnik preprost in funkcionalnost majhna.

Neposredno primerjavo hibridnih in izvirnih Android aplikacij bom predstavil v tabeli[10]:

Oba pristopa razvijanja aplikacije imata svoje prednosti. Največje prednosti, ki jih prinese hibridna aplikacija so hitrejši čas razvoja ter podpora različnih platform. Za razvoj aplikacije pa smo se v podjetju odločili za izvirni pristop, saj smo želeli čim boljšo uporabniško izkušnjo in dobro orodje

za razhroščevanje, saj mora aplikacija delati v vsem v popolnosti. Zaradi tega pristopa pa je bilo potrebno implementirati spletne storitve na strežniškem delu, kar je povečalo čas razvoja celotne aplikacije.

| | **Izvirne aplikacije** | **Hibridne aplikacije** |
|---|---|---|
| Hitrost razvoja | Počasneje pri manj kompleksnih aplikacijah, potreben ločen razvoj za vsako platformo posebej | Hitrejše pri manj kompleksnih projektih, vendar je zahtevnost večja, ko želimo uporabiti tehnologije, ki presegajo zmožnosti pospeševalnikov |
| Uporabniška izkušnja | Aplikacije so bolj odzivne in tekoče ter nimajo občutka, da se vseskozi nekaj nalaga. Kliki, drsniki in animacije so bolj tekoče in lepše. | Aplikacije včasih niso odzivne, kliki se ne beležijo, animacije niso tekoče. |
| Vzdrževanje | Spremembe je potrebno za vsako platformo vzdrževati posebej | HTML in CSS sta lažja za vzdrževanje, funkcionalnosti ni potrebno popravljati za vsako platformo posebej. |
| Omejitve | Aplikacije imajo dostop do vseh knjižic in vse funkcionalnosti, ki jih zagotavlja Android. | Aplikacije so omejene na tehnologijo HTML, uporaba različnih gradnikov ni omogočena. |
| Varnost | Za varnost je potrebno poskrbeti znotraj aplikacije. | Za varnost je potrebno poskrbeti tako na spletnih straneh kot na strežniškem delu. |
| Razhroščevanje | Vgrajeni razhroščevalnik je boljši, bolj pregleden in funkcionalen. | Razhroščevati je potrebno tako kot na spletnih straneh. |

Tabela 4.4: Primerjava izvirnih in hibridnih Android aplikacij.

# Poglavje 5

# Spletna blagajna

## 5.1 Kaj je spletna blagajna

Spletna blagajna je celovita blagajniška storitev primerna za vsa podjetja, katera morajo za svojo prodajo izdati račun. Storitev je bila razvita v podjetju Blocklogic d.o.o. in je namenjena podjetjem, ki za plačilo svojih storitev in blaga sprejemajo gotovino ali poslujejo s karticami preko POS-terminala. Namenjena je tudi podjetjem, ki za svoje storitve in blago sprejemajo plačilo preko ponudnika e-plačevanja Paypal in tistim, ki sprejemajo plačilo v obliki digitalne valute Bitcoin. Spletna aplikacija je bila razvita predvsem za pravne osebe, ki bodo morale zaradi prehoda v nov sistem, pri katerem bodo morale biti blagajne povezane z davčnim regulatorjem, zamenjati trenutno rešitev za izdajanje računov. Sem spadajo podjetja, ki za izdajanje računov uporabljajo zastarele programske rešitve, ter tisti, ki so do sedaj uporabljali paragonske bloke. Rešitev je primerna tudi za tiste, ki morajo izdajati račune na terenu, saj je Spletna blagajna storitev v oblaku in omogoča dostop od koderkoli preko enostavnih uporabniških vmesnikov. Prav tako je Spletna blagajna rešitev, ki je cenovno ugodna, saj je najcenejši paket brezplačen. Zato je primerna tudi za tiste, ki bi za svoje storitve radi odšteli manj ali pravzaprav nič.

Spletna blagajna je Point-of-sale (POS) rešitev v oblaku. Je program, ki

deluje na različnih platformah. Primarno je bil razvit za uporabo v spletnem brskalniku ali na operacijskem sistemu Android. Je računalniška zamenjava za blagajne, ki omogoča izdajanje računov v skladu z zakonodajo. Uporablja se, ko hoče kupec izvesti plačilo prodajalcu za izmenjavo blaga ali storitve. V tem koraku prodaje, mora prodajalec ponuditi predračun oziroma izračunati ceno kupcu. Ko se kupec odloči za nakup, mora sistem običajno izdati tudi račun. Spletna blagajna omogoča tudi vodenje inventure.

## 5.2 Uporaba spletne blagajne

Uporabnik se registrira na spletno blagajno preko spletne strani spletna-blagajna.si ali preko android aplikacije. Ko se uporabnik registrira, lahko doda svoje podjetje. Z vnosom svojega podjetja pa lahko uporabnik začne z uporabo aplikacije. Pred izdajanjem računov mora uporabnik dodati kategorije, produkte ter nastaviti svojo blagajno za izdajanje računov. Ko uporabnik konfigurira vse potrebno lahko začne uporabljati aplikacijo za izdajanje računov. Uporabniški vmesnik je prijazen ter omogoča hitro vnašanje artiklov. Artikli se lahko dodajajo tudi preko čitalca črtnih kod. Ko prodajalec vnese vse izbrane artikle, lahko zaključi račun. Ob zaključku računa prodajalec izbere še vrsto plačila med katera spadajo: gotovina, plačilna kartica, Paypal in možnost plačila z digitalno valuto Bitcoin. Za možnost plačila s kartico mora imeti prodajalec ločen terminal POS, katerega mu običajno zagotovi banka. Ko je vrsta plačila izbrana. se iztiska račun ali pa se račun pošlje preko e-pošte.

## 5.3 Obstoječe alternative na trgu

Na trgu je že nekaj podjetij, ki ponujajo podobne rešitve, vendar se v nekaterih pogledih razlikujejo od aplikacije Spletna blagajna. V tujini lahko zasledimo aplikacije kot so Vend, Square, Shopify in druge. Na domačem trgu pa se je ne dolgo nazaj pojavilo nekaj podjetji s konkurenčnimi rešitvami. Med

njimi lahko najdemo aplikacijo Si.blagajna, ki jo je razvilo podjetje Si.mobil, rešitev pa je razvilo tudi podjetje Telekom.

Vse te aplikacije ponujajo celovito blagajniško rešitev, vendar se med njimi pojavljajo tudi razlike. Vend je rešitev za uporabnike, ki so za delovanje storitev pripravljeni odšteti več denarja. Za njihove storitve boste odšteli od 59 pa vse do 169 dolarjev mesečno, odvisno od zahtev uporabe.

Vend za svoje delovanje potrebuje brskalnik ali tablični računalnik iPad, podpore za Android pa trenutno nima. Square je trenutno najbolj priljubljena rešitev za prenosno blagajno. Z njo boste lahko izdajali račune tako na operacijskem sistemu iOS, kot tudi na Android, vendar za razliko od Spletne blagajne nima podpore za brskalnike. Za uporabo same aplikacije Square ni potrebno odšteti nič, zaračunajo pa vam provizijo od 2.75% do 3.5% za vsako plačilo s kartico. Kot vidimo se Spletna blagajna od teh dveh razlikuje po naboru platform za uporabo, saj za delovanje aplikacije potrebujemo spletni brskalnik ali napravo z nameščenim Android sistemom. Prav tako se razlikuje po ceni, saj je najcenejši paket spletne blagajne brezplačen ter ne zaračunava nobenih dodatnih provizij. Prav tako je v Spletni blagajni sistem za sprejemanje digitalne valute Bitcoin.

Podjetji Si.mobil in Telekom sta opazili priložnost na področju mobilnih blagajn, ter tako izdali svoje rešitve. Si.mobilova aplikacija Si.blagjna za svoje delovanje potrebuje operacijski sistem Windows (od XP dalje) ali Android. Mesečna naročnina znaša 14,99 evrov. Telekomova rešitev pa za svojo uporabo potrebuje napravo Android, ostali operacijski sistemi pa trenutno niso podprti. Za uporabo boste odšteli od 13,30 evrov dalje. Spletna blagajna se od zgoraj navedenih rešitev torej ponovno razlikuje po ceni ter po podpori platform.

## 5.4 Brezplačna spletna blagajna

Ideja brezplačne spletne blagajne se je rodila v podjetju po tem, ko se je podjetje udeležilo startup pospeševalnika v Bolgariji. Tam so hoteli od nas

izvedeti po čem se razlikujemo od ostalih ter kakšna je naša ključna prednost. Ker odgovora na to vprašanje nismo imeli, smo se v Slovenijo vrnili praznih rok. Na poti nazaj pa se je rodila ideja, da bi lahko na izpisane račune vključevali reklame. Tako bi podjetje Blocklogic d.o.o od vsakega iztiskanega računa lahko prejelo plačilo za prikaz reklame. Če bi se uporabnik odločil za to izbiro, mu ne bi bilo potrebno plačevati mesečne naročnine, v nasprotnem primeru pa bi se lahko odločil za drug paket, ki tega ne bi vključeval. Ti paketi pa bi bili prav tako ugodni, saj bi mesečna naročnina znašala okoli 7 evrov.

# Poglavje 6

# Arhitektura aplikacije

Arhitektura aplikacije je sestavljena iz dveh delov: mobilne aplikacije in strežnika. Strežnik je sestavljen iz PostgreSQL podatkovne povezave, spletnega dela, ki omogoča uporabnikom dostop preko brskalnika ter nabora APIjev preko katerih dostopajo mobilne aplikacije. Mobilna aplikacija pa je sestavljena iz lokalne SQLite baze ter različnih uporabniških vmesnikov preko katere uporabnik dostopa do storitev.

## 6.1 Arhitektura spletne aplikacije

Spletna aplikacija je razdeljena na več delov. V njih lahko najdemo tako imenovane aplikacije, ki skrbijo za uporabnike, konfiguracijo, tiskanje, celotno upravljanje podatkov namenjeni prodaji izdelkov in storite ter drugo. Glavni deli do katerih je mobilna aplikacija dostopala so:

- aplikacija blusers, ki je skrbel za upravljanje z uporabniki.
- aplikacija mobile, v katerem se nahaja nabor APIjev za upravljanje s podatki.
- aplikacija sync, ki je skrbel za kontrole različic podatkovne baze ter ažurnost podatkov.

Za delovanje aplikacije skrbita še datoteki settings.py in urls.py.

### 6.1.1 Aplikacija blusers

Aplikacija blusers (krajšava za BlockLogic users) je paket, ki skrbi za registracijo in avtentikacijo uporabnikov. V njem je mogoče zaslediti funkcije, ki skrbijo za registracijo in avtentikacijo uporabnikov v sistem. Te funkcije omogočajo različne načine prijav in registracij (preko obrazcev, Google prijava) ter preverjajo pravilnost podatkov. V tej aplikaciji je mogoče zaslediti tudi funkcijo, ki avtomatično pošlje e-sporočilo za aktivacijo uporabnika.

### 6.1.2 Aplikacija mobile

Aplikacija mobile je paket v katerem se nahajajo vsi APIji preko katerih dostopa mobilna aplikacija. Tako je mogoče v njej zaslediti funkcije, ki skrbijo za vnos, urejanje in brisanje podatkov (produktov, kontaktov, davkov, kategorij,...). Te funkcije vseskozi uporabljajo kodo, ki je bila prvotno napisana za spletni del aplikacije. Koda se zaradi tega ne podvaja, s tem pa sem se izognil večkratnemu popravljanju istih funkcionalnosti ob spremembah na strežniku.

### 6.1.3 Aplikacija sync

Aplikacija sync skrbi za ažurnost podatkov. Preko prožilcev na spremembah podatkov v strežniški podatkovni bazi vodi ločeno tabelo, ki uporabnikom omogča, da so njihovi podatki ažurni. V tej aplikaciji je mogoče zaslediti signale, ki naredijo za vsako spremembo podatka v podatkovni bazi nov zapis v ločeni tabeli.

### 6.1.4 Datoteka settings.py

Djangotova datoteka settings.py vsebuje nastavitve strežnika. Tu so definirane osnovne nastavitve podatkovne baze. V tej datoteki sem definiral lokalno podatkovno bazo, saj sem aplikacijo razhroščeval na svojem računalniku. Ker je Django zelo zmogljivo orodje, zna podatkovno bazo kreirati sam, upo-

rabi pa kar objekte, ki so napisani v jeziku Python. Prav tako je v tej datoteki potrebno nastaviti vse aplikacije, ki jih bo strežnik uporabljal. Tako je potrebno dodati npr. aplikacijo mobile, če želimo dostopati do teh APIjev.

### 6.1.5 Datoteka urls.py

Datoteka urls.py v okolju Django definira vse možne urlje preko katerih je možno dostopati do storitev spletne aplikacije. Vsak zapis je v tej datoteki ločen z dveh ključnih vrednosti: ime urlja ter funkcija, ki se izvede po tem, ko vzpostavimo povezavo s tem urljem.

```
url(r_company + r '/manage/json/category/add/?$', category.mobile_add_category, name = 'add_category'),
```

Slika 6.1: Definiran url v okolju Django.

## 6.2 Arhitektura mobilne aplikacije

Ko pišemo Android aplikacijo postane naša koda velikokrat obširna, zato jo je smiselno razdeliti po delih, ki se imenujejo paketi. Zaradi večje preglednosti in funkcionalnosti je aplikacija Spletna blagajna razdeljena v več paketov. V njih je mogoče zaslediti pakete, ki skrbijo za upravljanje s podatkovno bazo, izmenjavo podatkov, avtentikacijo ter druge. Ker se v diplomski nalogi osredotočam na vpeljavo Android aplikacije v okolje Django, si bomo podrobneje ogledal le del kode, ki skrbi za uspešno komuniciranje med njima. Sem spadajo paketi:

- paket net.blocklogic.JSON v katerem so napisane funkcije za izemnjavo podatkov med strežnikom in mobilno aplikacijo,
- paket net.blocklogic.pos.sqlite, ki skrbi za kreiranje in spremembe lokalne podatkovne baze ter ažurnost podatkov,

- paket net.blocklogic.pos.product, ki bo uporabljen kot primer za vnos ter urejanje podatka.

Poleg tega je mobilna aplikacija razdeljena še na druge dele:

- mapa res, v kateri se nahajajo vsi grafični izgledi aplikacije,
- mapa gen, ki vsebuje avtomatsko generirane datoteke,
- mapa lib, ki vsebuje zunanje knjižnice, ki niso del Androidovih,
- datoteka AndroidManifest.xml, v kateri so definirane vse aktivnosti aplikacije, razna dovoljenja, ki jih aplikacija potrebuje (dostop do fotoaparata, bluetootha, pisanja na spomin,...) ter katero programsko opremo podpira naša aplikacija.

Podrobneje si bomo pogledali še datoteko AndroidManifest.xml ter paket net.blocklogic.pos, v kateri so definirani razni pripomočki, ki omogočajo, da se v aplikaciji koda ni podvajala.

### 6.2.1 Paket net.blocklogic.JSON

Paket skrbi za povezovanje s spletno aplikacijo ter vračanjem odgovora, ki ga vrne strežnik. V njem je možno zaslediti funkcije, ki uspešno skrbijo za vzpostavitev povezave s strežnikom ter vračanjem njegovega odgovora. Prav tako je mogoče zaslediti funkcijo, ki iz strežniškega odgovora izluščijo JSON objekt ali JSON niz, odvisno od tega kaj pričakujemo.

### 6.2.2 Paket net.blocklogic.pos.sqlite

Paket skrbi za upravljanje s podatkovno bazo SQLite, ki se nahaja lokalno na Android napravi. V njej se nahaja datoteka WebPosSQLite, ki je izpeljanka objekta SQLiteOpenHelper, ki skrbi za kreiranje ter urejanje podatkov na lokalni podatkovni bazi. V naši datoteki pa lahko najprej zasledimo funkcijo, ki kreira podatkovno bazo, ki se po strukturi približa spletni podatkovni bazi.

V naši datoteki je možno zaslediti funkcijo, ki skrbi za kakršnekoli posodobitve podatkovne baze ob spremembi tabel. Poleg tega pa lahko zasledimo vse funkcije za nove vnose v tabele ter iskanje zapisov po tabelah.

### 6.2.3 Paket net.blocklogic.pos.product

Ta paket nam bo predstavljal primer komunikacije med Android napravo in strežnikom. Podobnih paketov za upravljanje s podatki je več, razlikujejo pa se po strukturi objekta ter prikazovanjem podatkov na sami napravi. V tem paketu lahko zasledimo objekt Product, ki ima vse spremenljivke podane tako, da se čim bolj približajo podatkovnim tipom, ki se nahajajo na strežniški podatkovni bazi istega objekta. Poleg tega lahko v tem paketu zasledimo adapter, ki skrbi za prikazovanje seznama produktov. Za čim boljšo uporabniško izkušnjo, pa lahko v tem paketu zasledimo še različne Android gradnike, ki uspešno upravljajo to delo.

### 6.2.4 Paket net.blocklogic.pos

V tem paketu se nahajajo razni pripomočki, ki omogočajo, da se koda v aplikaciji ni podvajala. Tako lahko najdemo dialog, ki se izpiše v primeru napake, globalne spremenljivke, datoteko ki upravlja s shranjevanjem nastavitev lokalno, datoteko z vsesplošnimi nastavitvami ter drugo. Bolj zanimivi sta datoteki MySharedPreferences.java, ki skrbi, da so podatki, ki so shranjeni lokalno, varni in kriptirani. V datoteki WebPosApplication.java pa se nahajajo vsi URL naslovi do katerih aplikacija dostopa.

### 6.2.5 Datoteka AndroidManifest.xml

V tej datoteki sem definiral vse aktivnosti, ki so se uporabljale skozi življenjski cikel aplikacije. Tu sem določil tudi najmanjšo različico sistema Android, ki pa je 2.3. S tem sem pokril 96% vseh Android naprav. Poleg tega sem definiral tudi dovoljenja do katerih lahko aplikacija dostopa. Med njimi so dostop do interneta, dostop do tehnologije Bluetooth, upravljanje z uporabniki na

sistemu, dovoljenje za avtentikacijo uporabnikov ter dostop do fotoaparata. V tej datoteki se nahaja naslov aplikacije ter njena zagonska ikona.

# Poglavje 7

# Razvoj aplikacije

V tem poglavju si bomo podrobneje pogledali vse tehnologije, ki so bile uporabljene pri izdelavi aplikacije. Podrobneje bom predstavil povezovanje aplikacije s spletnim strežnikom ter v tem okviru predstavil registracijo in avtentikacijo uporabnikov. Nato si bomo pogledali kako delujejo funkcije, ki skrbijo za ažurnost podatkov. V tretjem delu bom predstavil tehnologije, ki so skrbele za izpisovanje računov. Na koncu pa bomo pogledali en primer uporabe aplikacije.

Na visokošolskem študiju računalništva in informatike smo se lahko spoznali s tehnologijami za razvoj mobilnih aplikacij, zato sem imel odlično izhodišče za razvoj naše aplikacije. Malo težav mi je od začetka delal strežniški del aplikacije, saj z okoljem Django še nisem bil seznanjen. Na srečo pa ima Django dober uvodni vodič in obširno dokumentacijo, ki mi je delo olajšala.

Ideja za aplikacijo je dobil direktor podjetja, ki je razvoj aplikacije tudi financiral. Ker je velikokrat videl, da imajo trgovine, tržnice in lokali zastarel in nepregleden sistem za prodajo, je prišel na idejo, da bi lahko to izboljšal. Želel je pregleden, preprost in hiter sistem za prodajo, ki je dostopen z več naprav. Želel je tudi sistem, ki ima vse podatke shranjene v oblaku.

Za grafično podobo aplikacije je skrbel grafični oblikovalec. Potem ko sem mu podal svoje zahteve in pokazal približno postavitev gradnikov, je izrisal končno podobo vsake aktivnosti posebej. Aktivnosti sem nato oblikoval po

njegovih skicah.

Mobilna in spletna aplikacija potrebujeta za svoje delovanje dostop do interneta. Tako potrebuje mobilna aplikacija dostop do interneta preko tehnologije Wifi ali GPRS. To pa uporabniku velikokrat predstavlja strošek, povezava GPRS pa je včasih tudi počasna. Zato ima mobilna aplikacija ločeno lokalno bazo SQLite, v kateri so podatki ažurni. Tako se izognemo nepotrebnemu vnovičnemu prenašanju vseh podatkov.

## 7.1 Povezovanje androidne aplikacije s spletno

V tem delu bom predstavil primer kako se mobilna aplikacija povezuje s spletno. Opisal bom kako mobilna aplikacija uspešno generira klienta za povezovanje s spletno aplikacijo ter mu doda vse potrebne podatke, s katerimi spletna aplikacija operira. V tem delu bom opisal tudi kako mobilna aplikacija predela podatke, ki mu jih strežnik vrne. Nato bom opisal še kako povezovanje aplikacije deluje na spletnem delu. V tem delu bom predstavil dekoraterje, ki so potrebni za uspešno izvršitev funkcije.

### 7.1.1 Mobilni del

Pri mobilnem delu sem najprej sprogramiral razred, ki vsebuje funkcije za generiranje klienta za povezavo s strežnikom, funkcijo, ki je ta klient uporabila, ter mu dodala uspešne atribute in podatke, ki jih je nato strežnik obdelal ter funkcijo, ki prebere s strežnika odgovor v obliki zapisa JSON. Klient je poskrbel, da se lahko aplikacija varno povezovala preko protokola SSL brez strežniškega certifikata, saj ta trenutno še ne obstaja.

Funkcija, ki skrbi za povezovanje z aplikacijo s strežnikom ta klient generira, ter mu nato doda v primeru, da je avtentikacija potrebna, žeton. Žeton ni potreben le v primeru, ko se uporabnik prijavlja z uporabniškim imenom in geslom, takrat pa sta za avtentikacijo potrebna ta dva podatka. Po tem funk-

cija doda še potrebne podatke, ki jih strežnik razčleni za uspešno izvršitev. Ti podatki so npr. podatki o produktu, ki ga shranjujemo. Podatki se v večini primerih zapisani v obliki JSON, v enem primeru pa mobilna aplikacija pošlje podatke v obliki XML. Ta primer se pojavi takrat, ko se uporabnik želi registrirati, saj strežnik v tem primeru preveri pravilnost podatkov v obliki forme, ki je v okolju Django izbran kot privzet način razčlenjevanja podatkov. V razredu, ki ga trenutno opisujem pa se pojavi še funkcija, ki iz odgovora s strežnika izlušči zapis JSON.

```
1  public HttpResponse post(String url, String token, JSONObject jsonData) {
2
3      HttpClient httpClient = getNewHttpClient();
4      HttpParams params = httpClient.getParams();
5      params.setParameter("http.protocol.handle-redirects", false);
6      HttpResponse response = null;
7      URI address = null;
8
9      try {
10         address = new URI(url);
11     } catch (URISyntaxException e1) {
12         e1.printStackTrace();
13     }
14
15     HttpPost httppost = new HttpPost(address);
16     // httppost.addHeader("X-Requested-With", "XMLHttpRequest");
17     httppost.setParams(params);
18
19     if (token != null && token.length() > 0) {
20         httppost.addHeader("Authorization", "Token " + token);
21     }
22     if (jsonData != null) {
23         List < NameValuePair > postParams = new ArrayList < NameValuePair > (1);
24         postParams.add(new BasicNameValuePair("data", jsonData.toString()));
25         try {
26             httppost.setEntity(new UrlEncodedFormEntity(postParams, "UTF-8"));
27         } catch (UnsupportedEncodingException e) {
28             e.printStackTrace();
29         }
30     }
31     try {
32         response = httpClient.execute(httppost);
33     } catch (ClientProtocolException e) {
34         e.printStackTrace();
35     } catch (IOException e) {
36         e.printStackTrace();
37     }
38
39     return response;
40 }
41
```

Slika 7.1: Povezovanje mobilne aplikacije s spletno.

```
public JSONObject GetObjectFromResponse(HttpResponse response) {

    JSONObject jsonO = null;
    BufferedReader reader;

    String json;
    try {
        if (response != null) {
            reader = new BufferedReader(new InputStreamReader(response.getEntity().getContent(), "UTF-8"), 8192);

            json = reader.readLine();
            jsonO = new JSONObject(json);

        }
    } catch (UnsupportedEncodingException e1) {
        e1.printStackTrace();
    } catch (IllegalStateException e1) {
        e1.printStackTrace();
    } catch (IOException e1) {
        e1.printStackTrace();
    } catch (JSONException e) {
        e.printStackTrace();
    }

    return jsonO;

}
```

Slika 7.2: Luščenje JSON objekta iz strežniškega odgovora.

Za povezovanje mobilne aplikacije s spletno skrbijo asinhrone funkcije. Te v prvem delu izluščijo podatke iz aplikacije, ki se pošljejo spletni, ta pa podatke obdela. Spletna aplikacija mobilni odgovori s statusom izvršitve ali napake. V primeru napake mobilna aplikacija izriše dialog s sporočilom napake, v primeru izvršitve pa aplikacija nadaljuje svoj proces.

### 7.1.2 Spletni del

Pri spletnem delu sem napisal vse funkcije, ki so izluščile podatke, ki jih je spletna aplikacija dobila od mobilne. Te funkcije so nato klicale druge, katere so bile napisane s strani sodelavcev, te pa so podatke pravilno obdelale in shranile. Na koncu so moje funkcije odgovorile mobilni aplikaciji z ustreznim statusom obdelave ter po potrebi še z aktualnimi podatki.

Vse te funkcije sem ovil z dekoraterjem `api_view`, ki je del ogrodja Django REST framework. Ta dekorator se uporablja, z namenom da uporabimo poglede (angl. views), ki so del tega ogrodja in ne tistih, ki so del ogrodja Django. Ti pogledi se razlikujejo po načinu predelave zahtev (angl. requests) in odgovorov (angl. response). Glavna razlika teh pogledov je, da okolje Django vrača odgovore s podatki in jim poda vizualno obliko, medtem ko

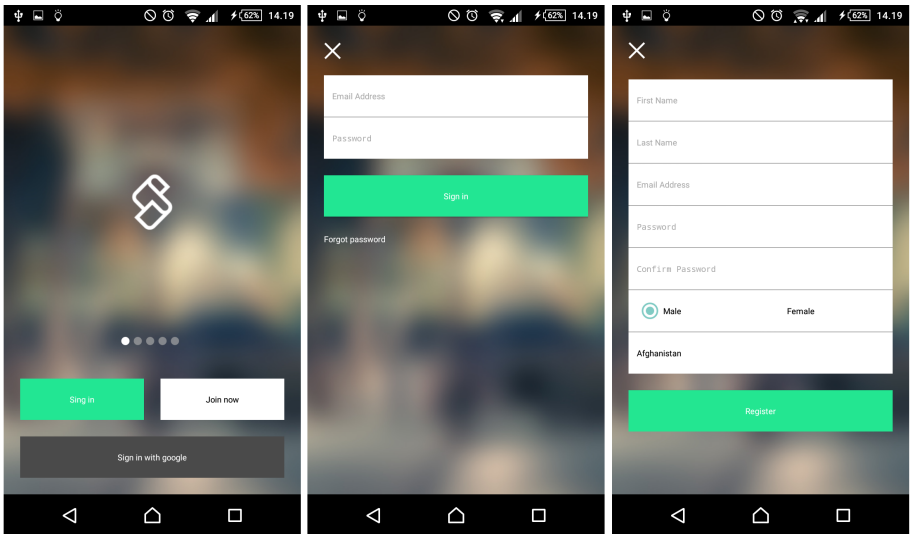
REST Framework vrača le surove podatke v bolj primitivnih oblikah. Tako lahko podatke v odgovoru dodamo v obliki slovarja (angl. dictionary), REST Framework pa se bo obnašal, kot da so ti podatki v obliki JSON. Poleg tega vsebuje REST Framework še dekorator `permmision_classes`, kateri preverja, če je uporabnik avtenticiran. Iz glave poizvedbe prebere žeton, ki mu ga je mobilna aplikacija poslala, ter nato iz tega žetona ugotovi za katerega uporabnika gre ter preveri njegove pravice za dostop do podatkov.

```
@api_view(['POST', 'GET'])
@permission_classes((IsAuthenticated, ))
def mobile_JSON_categories(request, company_id):
    try:
    c = Company.objects.get(id = company_id)
except Company.DoesNotExist:
    return JsonError(_("Company does not exist"))

# permissions
if not has_permission(request.user, c, 'category', 'view'):
    return JsonError(_("You have no permission to view categories"))
data = get_all_categories(c, android = True, no_category = True)

# return all categories ' data in JSON format
return JsonOk(extra = data, safe = False)
```

Slika 7.3: Primer funkcije v Djangu, ki vsebuje potrebne dekoratorje, ter vrne seznam produktov.

## 7.2 Avtentikacija uporabnika

Za uporabo aplikacije je potrebna registracija. To lahko uporabnik naredi na dva načina, preko registracijskega obrazca ali prijave preko storitve google. Za registracijo so potrebni naslednji podatki: ime, priimek, e-poštni naslov, spol, država, geslo ter potrditev gesla. Ko uporabnik bodisi vnese podatke v ta obrazec, bodisi se prijavi preko Googlove storitve za prijavo, se v strežniku doda nov zapis v tabeli uporabnikov. Po tem koraku lahko uporabnik začne uporabljati aplikacijo.




Slika 7.4: Prijava in registracija v sistem.

### 7.2.1 Mobilni del

Ko se uporabnik odloči za prvo prijavo v aplikacijo ima naslednje možnosti. Če se je uporabnik že registriral preko obrazca, se lahko v aplikacijo vpiše s svojim uporabniškim imenom in geslom. Če uporabnik registracije še ni izvedel, se lahko v aplikacijo registrira preko obrazca ali uporabi Googlovo storitev za prijavo. V kolikor izbere slednje, strežnik preveri ali je uporabnik že v tabeli uporabnikov. Če uporabnik še ne obstaja, ga strežnik doda. Ko je uporabnik uspešno zabeležen v bazi, se lahko prijavi v aplikacijo. Za prijavo v aplikacijo bo moral uporabnik izbrati vedno isti način, kot ga je uporabil pri registraciji.

Ob prijavi v sistem strežnik ustvari žeton, ki skrbi za avtentikacijo uporabnika. Od tega trenutka dalje se za vsako povezavo, ki jo mobilna aplikacija naredi s strežnikom v glavi sporočila HTTP doda ta žeton. Tako ni potrebno za vsako nadaljnjo povezavo vnovično pošiljati uporabniškega imena in gesla za preverjanje identifikacije uporabnika. Prav tako ni priporočljivo shranjevati gesla na lokalni pomnilnik naprave, saj je to sporno z varno-

stnega vidika. V trenutku, ko se uporabnik prijavi v sistem, se tako v lokalni pomnilnik shranita uporabniško ime ter žeton, ki pa sta zaradi varnostnih razlogov še dodatno kriptirana. Ko sta v aplikaciji shranjena ta dva podatka, se ob vsakem naslednjem zagonu aplikacija samodejno vpiše uporabnika, ki je nazadnje uporabljal to aplikacijo.

### Prijava v aplikacijo

Če uporabnik izbere način prijave z uporabniškim imenom in geslom je s strani programerja to preprost primer. Aplikacija pošlje uporabniško ime in geslo na strežnik, strežnik odgovori z avtentikacijskim žetom, nato ga ta shrani v lokalni pomnilnik, uporabnik pa lahko začne z uporabljanjem aplikacije.

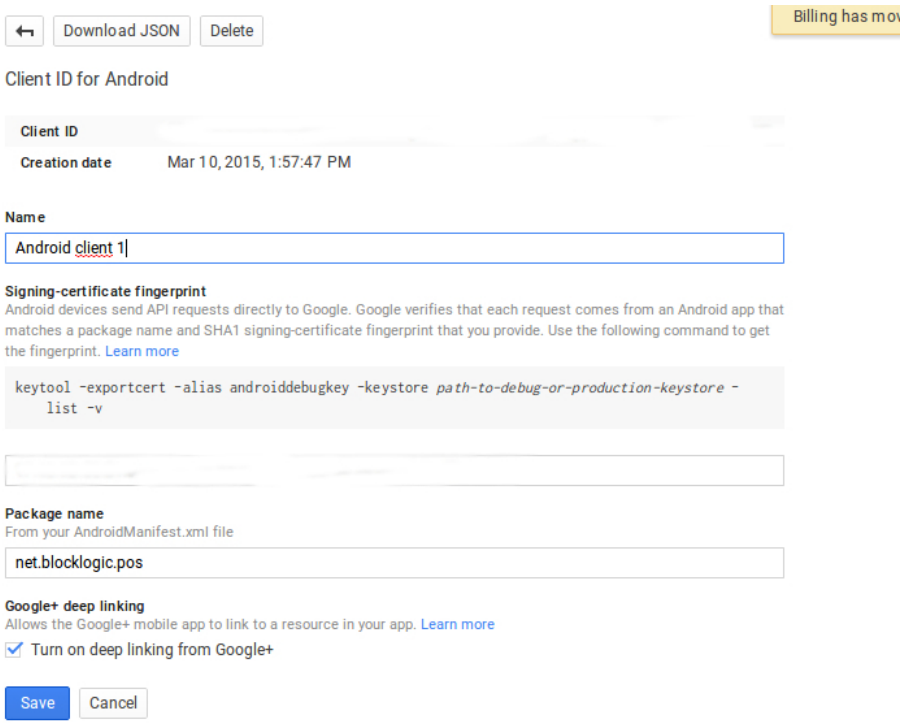
V primeru prijave preko Googlove storitve za prijavo pa mora programer poseči po dokumentaciji za razvijalce Android aplikacije. V prvem koraku je potrebno nastaviti storitev, ki povezuje aplikacijo z Googlovo storitvijo. V tem koraku je potrebno določiti tudi vse obsege (angl. Scopes), ki jih bomo od uporabnika želeli. Med te obsege spadajo npr. poštni naslov, ime, priimek ter drugi. Seveda pa morajo biti obsegi enaki na strežniku in mobilni aplikaciji, saj nam v nasprotnem primeru Googlova storitev zavrne dostop do teh podatkov.

```
1  mGoogleApiClient = new GoogleApiClient.Builder(this)
2      .addConnectionCallbacks(this)
3      .addOnConnectionFailedListener(this)
4      .addApi(Plus.API)
5      .addScope(Plus.SCOPE_PLUS_LOGIN)
6      .addScope(new Scope("https://www.googleapis.com/auth/userinfo.email"))
7      .addScope(new Scope("https://www.googleapis.com/auth/plus.profile.emails.read"))
8      .addScope(new Scope("https://www.googleapis.com/auth/userinfo.profile"))
9      .build();
10
```

Slika 7.5: Generiranje obsegov v Androidu.

Za uspešen dostop do storitev pa je potrebno definirati tudi dostop do APIjev v Googlovi konzoli. Tam moramo navesti podpis aplikacije (podpis

aplikacije zagotavlja, da do teh APIjev zagotovo dostopa naša aplikacija) ter paket aplikacije, v katerem dostopamo do aplikacije.




Slika 7.6: Generiranje Google APIjev v konzoli za Google prijavo.

Ko imamo definirano vse potrebno, se lahko uporabnik prijavi v naš sistem preko te storitve. Ob pritisku na gumb se mu izpiše novo okno, kjer je navedeno katere podatke želimo dobiti od uporabnika. Če uporabnik dovoli aplikaciji dostop do teh podatkov, se ta poveže z Googlovim strežnikom, ta pa mu generira žeton, ki določa identifikacijo uporabnika. Ta žeton je podoben tistemu v naši aplikaciji, vendar pa ne gre za istega, saj se žeton, ki ga ustvari Google uporablja izključno za pridobitev podatkov uporabnika in ne za dostop do naše aplikacije. Ko naša mobilna aplikacija dobi Googlov žeton, ga pošlje na naš strežnik. Strežnik preko žetona preveri za katerega uporabnika gre, v primeru da še ne obstaja pa ga doda v tabelo uporabnikov.

Ob uspešni prijavi strežnik vrne enak odgovor kot pri prijavi z uporabniškim imenom in geslom, uporabnik pa lahko nato uporablja aplikacijo.

#### Shranjevanje uporabnika in žetona

Ob prijavi v aplikacijo se uporabnik doda v Androidov pripomoček za upravljanje z uporabniki (angl. AccountManager). Ta skrbi za shranjevanje podatkov o uporabnikih, eden izmed teh podatkov pa je žeton. Androidov pripomoček omogoča varno shranjevanje teh podatkov. Prav tako omogoča dostop do storitev medtem ko se aplikacija ne izvaja, saj lahko takrat posodabljamo npr. podatkovno bazo.

Za samodejno vpisovanje aplikacije je lokalno shranjeno uporabniško ime uporabnika, ki je zadnji uporabljal to aplikacijo. Ko se aplikacija zažene, pripomoček za upravljanje z uporabniki najde uporabnika ter vrne žeton, s katerim nato uporabnik dostopa do storitev.

### 7.2.2 Strežniški del

Pri implementaciji strežniškega dela je bilo potrebno napisati funkcijo, ki preveri uporabniško ime in geslo ter uspešno generira žeton za nadaljnjo uporabo. Za lažjo implementacijo je skrbelo ogrodje Django REST Framework, ki ima te funkcionalnosti že vključene. V funkciji je bilo potrebno opredeliti vrste avtentikacij (angl. authentication classess), za kateri sem izbral TokenAuthentication - prijava z žetonom, ki ga generira strežnik, ter OAuth2Authentication - prijava z žetonom, ki ga generira Google. Uporabnik se avtenticira na tri načine. Na strežnik lahko pošlje uporabniško ime in geslo, pošlje žeton, ki ga je ustvarila naša spletna aplikacija ali z žetonom, ki ga je ustvaril Google. Funkcija preveri pravilnost podatkov, mobilni aplikaciji pa odgovori z našim žetonom, ki ga nato mobilna aplikacija uporablja za nadaljnjo preverjanje identitete uporabnika.

```
class ObtainAuthToken(APIView):
    throttle_classes = ()
    permission_classes = ()
    parser_classes = (parsers.FormParser, parsers.MultiPartParser, parsers.JSONParser,)
    renderer_classes = (renderers.JSONRenderer,)
    serializer_class = AuthTokenSerializer
    authentication_classes = (OAuth2Authentication, TokenAuthentication,)
    model = Token

    def post(self, request, backend):

        if backend == 'auth':
            if request.META and 'HTTP_AUTHORIZATION' in request.META:
                user = get_user(request)
            else:
                serializer = self.serializer_class(data=request.DATA)
                if serializer.is_valid():
                    user = serializer.object['user']
                else:
                    return JsonError(message=_("wrong credentials"))

        elif backend == "google-oauth2":
            d = google_login_or_register(request, mobile=True)
            if d['status'] == 'ok':
                user = BlocklogicUser.objects.get(id=d['user_id'])
            else:
                return JsonResponse(d)

        else:
            return JsonError(message=_("wrong login"))

        token, created = Token.objects.get_or_create(user=user)

        if user:
            user_credentials = get_user_credentials(user)
        else:
            return JsonError(_("User authentication failed."))

        companies = user.companies
        companies_list = [company_to_dict(user, i, android=True, with_config=True)
for i in companies]
        actions = get_actions(user)

        return JsonResponse({'token': token.key,
                             'user': user_credentials,
                             'actions': actions,
                             'companies': companies_list,
                             'status': "ok"})

obtain_auth_token = ObtainAuthToken.as_view()
```

Slika 7.7: Generiranje žetona in avtentikacija uporabnika.

### 7.2.3 Težave pri implementaciji prijave

Največji problem pri implementaciji prijave mi je delala prijava s storitvijo Google. Google s svojimi posodobitvami velikokrat spremeni način dostopa do strežnika, zato je bilo potrebno tudi velikokrat popravljati kodo. Prav tako je bilo potrebno paziti določanje obsegov, katere pokriva prijava s sistemom Google, saj so morali biti ti v skladu s strežniškimi. Android ima v knjižnicah

že v naprej definirane obsege, ki pa so se v manjših podrobnostih razlikovali od strežniških. Strežnik je tako zavračal prijavo s sistemom Google, dokler nisem na roke popravil obsege, identičnim tistim na strežniku. Poleg tega pa je bilo potrebno paziti tudi pri konzoli Google, saj je postopek generiranja ključev za dostop do storitev prijave drugačen, ko aplikacijo razhroščujemo, od tistega, ko aplikacijo izdamo za uporabo.

## 7.3 Lokalna podatkovna baza

Število podatkov, s katerimi uporabnik upravlja v aplikaciji je veliko. Vsi produkti, kontakti, kategorije in drugo so zapisi v tabelah na strežniku, vsak zapis pa predstavlja ceno in to je velikost. Mobilna aplikacija bi se lahko nenehno povezovala na strežnik in vnovično prenašala ažurne podatke, vendar bi to to prineslo številne slabe lastnosti. Uporabniku bi predstavljal strošek pri uporabi mobilnih podatkov, sam prenos pa bi zahteval čas, katerega pa ne želimo po nepotrebnem zapravljati. Slabe lastnosti pa bi prineslo tudi strežniku, saj bi za prenos podatkov moral ta nenehno brati iz podatkovne baze, ta proces pa bi obremenjeval procesor in disk ter porabljal pasovno širino interneta.

Vsemu temu sem se hotel izogniti in prišel do rešitve, da sem v mobilni aplikaciji ustvaril ločeno podatkovno bazo, v kateri so shranjeni ažurni podatki.

### 7.3.1 Ideja

Ideja je bila sprogramirati ločeno podatkovno bazo v mobilni aplikaciji, ki se po strukturi zelo približa tisti, ki je na strežniku. Razlikuje se po tem, da sem lahko v mobilni aplikaciji spustil nekaj polj in relacij, ki za samo funkcionalnost niso bili potrebni. V podatkovni bazi so shranjeni vsi podatki do katerih uporabnik dostopa, podatki pa so vedno ažurni.

### 7.3.2 Ažurnost podatkov

Največji del pri implementaciji lokalne podatkovne baze je predstavljal razvoj funkcionalnosti, ki je skrbela za ažurne podatke. Rešitev, ki sem si jo predstavljal, je bila naslednja. V mobilni aplikaciji je lokalna podatkovna baza, za njeno ažurnost pa je skrbela funkcija. Ta se je v prvem koraku povezala s strežnikom, strežnik pa ji je odgovoril z vsemi podatki, ki še niso ažurni. Tako je naša funkcija od strežnika dobila vsak nov zapis, spremembo, ali izbris podatkov, ki so se že zgodili, vendar še niso bili knjiženi v lokalni podatkovni bazi. Ko so podatki posodobljeni, lahko uporabnik nadaljuje z uporabo aplikacije.

Strežniški del rešitve pa je predstavljal funkcionalnost, ki je zagotavljala, da mobilni aplikaciji pošilja samo tiste podatke, ki niso ažurni. Za spremembe na podatkovni bazi je služila ločena tabela, v katero se shranjujejo podatki o vseh spremembah, ki se dogajajo na podatkovni bazi na strežniku. V ta namen sem ustvaril sprožilce (angl. trigger) na vseh tabelah, ki jih mobilna aplikacija uporabljala. Tako se za vsako spremembo na podatkovni bazi doda nov zapis v našo novo tabelo sprememb. Ker nismo želeli, da je naša tabela sprememb preobširna, pa smo funkcionalnost prilagodili tako, da je tabela vodila le zadnjih n-zapisov (n je število, trenutno nastavljeno na 2000, ki še ni bilo določeno, saj jo je za čim večjo funkcionalnost potrebno dobro izračunati).

Za kontrolo verzije skrbi števec, ki je shranjen lokalno na mobilni napravi. Ta števec predstavlja zadnjo spremembo po vrsti, ki je na mobilni napravi že ažurna. Mobilna aplikacija posreduje ta števec strežniku, ta pa preveri ali je od zadnje različice bilo več kot n sprememb. V primeru, da jih ni, strežnik vrne odgovor s podatki o vseh spremembah, drugače pa pošlje vse podatke, ki jih uporabnik potrebuje. Ko mobilna aplikacija dobi podatke, nastavi števec na zadnjo različico podatkovne baze.

### 7.3.3 Mobilni del

V mobilnem delu sem najprej ustvaril podatkovno bazo. Za upravljanje s podatkovno bazo je v Androidu namenjen razred SQLiteOpenHelper, za polno funkcionalnost pa je potrebno naš razred, v katerem smo definirali podatkovno bazo razširiti z njim. Ko sem imel ustvarjeno podlago, sem napisal vso SQL sintakso za ustvarjanje tabel ter relacij med njimi. Ta sintaksa pa se prvič sproži, ko želi uporabnik dostopati do podatkovne baze oziroma takrat, ko podatkovna baza ne obstaja. Ko sem imel podatkovno bazo postavljeno, sem napisal vso programsko kodo, ki je skrbela za dodajanje, urejanje in brisanje zapisov za posamezno tabelo. Poleg tega pa sem napisal še vse funkcije, ki so skrbele za poizvedbe v podatkovni bazi ter za prave kriterije vračale objekte, ki jih uporabnik mobilne aplikacije potrebuje.

```
1  public void addCategory(Category category) {
2      SQLiteDatabase db = this.getWritableDatabase();
3
4      ContentValues values = new ContentValues();
5
6      values.put(CATEGORY_KEY_ID, category.id);
7      if (category.parent_id > 0) values.put(CATEGORY_KEY_PARENT, category.parent_id);
8      else values.putNull(CATEGORY_KEY_PARENT);
9      if (category.description.length() > 0) values.put(CATEGORY_KEY_DESCRIPTION, category.description);
10     else values.putNull(CATEGORY_KEY_DESCRIPTION);
11     values.put(CATEGORY_KEY_NAME, category.name);
12
13     if (category.color != null) {
14         values.put(CATEGORY_KEY_COLOR, category.color);
15     } else {
16         values.putNull(CATEGORY_KEY_COLOR);
17     }
18
19     db.insertWithOnConflict(TABLE_CATEGORIES, null, values, SQLiteDatabase.CONFLICT_REPLACE);
20     db.close();
21 }
22
```

Slika 7.8: Funkcija za dodajanje kategorije v podatkovno bazo.

```
public Category getCategory(int id) {
    String query = "SELECT * FROM " + TABLE_CATEGORIES + " WHERE " + CATEGORY_KEY_ID + "=" + String.valueOf(id);
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(query, null);

    Category cat = null;
    if (cursor.moveToFirst()) {
        cat = new Category(cursor);
    }
    cursor.close();
    db.close();

    return cat;
}

```

Slika 7.9: Funkcija za pridobivanje kategorije iz podatkovne baze.

Ko je bila podlaga napisana sem se lotil funkcije, ki je ažurirala podatke. Ta se izvede v trenutku, ko uporabnik vstopi v aplikacijo. Če je kontrolna različica podatkovne baze enaka nič, ali če je od zadnjega ažuriranja prišlo več kot todo do 2000 sprememb, mobilna aplikacija od strežnika dobi vse podatke, ki jih potrebuje za funkcionalnost aplikacije. V nasprotnem primeru strežnik vrne odgovor samo s podatki, ki jih je potrebno ažurirati. Mobilna aplikacija preveri za kateri objekt gre ter dejanje, ki ga mora naredi z njim (ustvariti, urediti, zbrisati).

```
JSONObject j = json.getJSONObject("data");
version = j.getInt("version");
if (j.getBoolean("updated")) {
    Log.i("DATABASE", "DATABASE_UPDATED");
    onSynced(null);
    changeFragment(tf);
} else {
    /*
     * UPDATE DB
     */
    if (j.getBoolean("drop")) {
        db.TaxDropAndCreate();
        db.ProductsDropAndCreate();
        db.CategoryDropAndCreate();
        db.ContactsDropAndCreate();
        db.DiscountsDropAndCreate();
        db.RegisterDropAndCreate();

        JSONArray categories = j.getJSONArray("categories");

        for (int i = 0; i < category.length(); i++) {
                Category c = new Category(categories.getJSONObject(i), false);
                if (c != null) db.addCategory(c);

        }

    } else {

        JSONArray items = j.getJSONArray("items");

        for (int i = 0; i < items.length(); i++) {
            SyncItem si = new SyncItem(items.getJSONObject(i));

            else if (si.model.equalsIgnoreCase("category")) {
                if (si.action.equalsIgnoreCase("save")) {
                    Category c = new Category(si.item, false);
                    if (c != null) db.addCategory(c);
                } else {
                    db.removeCategory(si.object_id);
                }
            }

            db.setDBVersion(version);


    }
```

Slika 7.10: Del funkcije, ki preveri za kateri objekt gre in ga ažurira

Ko funkcija ažurira podatke v prvem koraku, preveri za kateri tip objekta gre. Med tipe objekta spadajo produkti, kategorije, kontakti, popusti, davki, nastavitve blagajne ter nastavitve podjetja. Ko funkcija ugotovi za kateri objekt gre, preveri kakšno dejanje mora narediti s tem objektom. V primeru, da je dejanje shranjevanje objekta, mobilna aplikacija kreira objekt iz oblike JSON, ki ga je prejel s strežnika ter ga nato doda v našo tabelo. V tabelo dodaja na način zamenjave v primeru, da objekt z istim enoličnim identifikatorjem (angl. ID) v naši tabeli že obstaja oziroma doda, če ta zapis še ne obstaja. Tako v primeru, da je bil objekt v času od prejšnjega ažuriranja urejen, mobilna aplikacija zamenja celoten objekt z ažurnim. V primeru, da je dejanje brisanje objekta, pa mobilna aplikacija iz oblike JSON, ki ga je

prejel s strežnika, prebere samo enolični identifikator, objekt poišče v lokalni podatkovni bazi ter ga izbriše.

### 7.3.4 Spletni del

Pri spletnem delu programiranja sem moral najprej ustvariti ločeno tabelo, v katero so se shranjevali opisi sprememb na podatkovni bazi. Ta tabela je sestavljena iz podjetja, v katerem se je sprememba naredila, za katero dejanje gre (shranjevanje, brisanje), za kateri objekt gre (kategorija, produkt, ipd.), enolični identifikator objekta ter kontrole verzije. Kontrola verzije se povečuje za vsako podjetje, saj je v mobilni aplikaciji važno, da so podatki ažurni glede na vsako podjetje.

```
1 class Sync(models.Model):
2     company = models.ForeignKey(Company)
3     action = models.CharField(max_length=20, null=False, blank=False)
4     model = models.CharField(max_length=20, null=False, blank=False)  # opisno, kateri model
5     object_id = models.IntegerField(null=False)
6     seq = models.IntegerField(null=False)  # za vsako firmo se zacne z 0 in se poveca z vsakim updejtom
7                                            # kateregakoli modela, ki mu sledimo s syncom
8
```

Slika 7.11: Kreiranje tabele, ki skrbi za ažuriranje podatkov.

Ko je bila tabela postavljena sem se lotil programiranja sprožilcev. Za sprožilce je v okolju Django poskrbljeno, implementacija pa je zaradi tega preprosta. Funkcija, ki se izvede v primeru spremembe v podatkovni bazi je potrebno opremiti z ustreznim dekoratorjem. Ta se imenuje receiver, poleg tega pa je potrebno definirati na katerih objektih se sprožilec izvede ter s kakšnim dejanju. Tako sem svojo funkciji določil dekoratorje za vse svoje objekte ter definiral, da se je funkcija sprožila po tem, ko je bil objekt shranjen ali brisan. To se v Django okolju definira tako, da se dekoratorju določi atributa `post_save` (po tem, ko je bil objekt shranjen) ter `post_delete` (po tem, ko je bil objekt izbrisan).

Naslednje stvar, ki sem jo moral napisati je bila funkcija, ki uspešno shrani objekt v našo novo tabelo ažurnih objektov. V prvem delu funkcija pridobi vse objekte, ki jih je potrebno ažurirati, saj nas zanima, kakšna je

zadnja kontrolna verzija naših objektov. Če je objektov več kot 2000, najbolj neaktualnega izbriše, saj sem se odločil, da lahko uporabnik mobilne aplikacije ažurira le zadnjih 2000 objektov, v nasprotnem primeru mora prenesti celoten nabor podatkov. Tako sem se izognil, da ne bi naša tabela ažurnih podatkov rasla v neskončnost. Naslednja stvar, ki jo funkcija naredi je, da preveri če obstaja objekt, ki ga želimo ažurirati med našimi zadnjimi 2000 ažurnimi objekti. V primeru, da obstaja, ga funkcija zamenja z bolj ažurnim, saj uporabnika, ki uporablja aplikacijo zanima samo zadnji najbolj ažuren podatek in ne vse spremembe, ki so se na njem dogajale. Objektu poveča kontrolo verzije ter ga shrani. V nasprotnem primeru, pa ažurni podatek zapiše kot nov objekt v tabelo ter mu poveča kontrolo verzije.

```
def signal_change(instance, created, action, **kwargs):
    from sync.models import Sync

    if hasattr(instance, 'company'):
        company = instance.company
    elif isinstance(instance, Company):
        company = instance
    else:
        # Should not happen
        return

    sync_objects = Sync.objects.only('seq')\
        .filter(company=company).order_by('-seq')


    # delete the last one so we don't have a huuuge db
    if len(sync_objects) > 2000:
        sync_objects[len(sync_objects)].delete()

    last_key = 0

    if len(sync_objects) > 0:
        last_key = sync_objects[0].seq

    try:
        object = Sync.objects.get(company=company, object_id=instance.id, model=instar
        object.seq = last_key+1
        object.action = action
        object.save()
    except Sync.DoesNotExist:
        sync = Sync(company=company,
                    action=action,
                    model=instance.__class__.__name__,
                    object_id=instance.id,
                    seq=last_key+1)
        sync.save()


@receiver(post_delete, sender=Category)
def set_serial_delete(instance, **kwargs):
    signal_change(instance, False, action='delete', **kwargs)


@receiver(post_save, sender=Category)
def set_serial_save(instance, created, **kwargs):
    signal_change(instance, created, action='save', **kwargs)
```

Slika 7.12: Prožilca in funkcija, ki se izvede ob prožilcu.

Zadnja stvar, ki sem jo sprogramiral v tem okviru je funkcija, s katero se je mobilna aplikacija povezovala, ta pa vrača podatke, ki na mobilni aplikaciji še niso ažurni. Za uspešno izvršitev funkcije mora mobilna aplikacija spletni poslati svojo zadnjo različico podatkovne baze. Spletna aplikacija nato primerja svojo različico podatkovne baze z uporabnikovo. V primeru, da je bila uporabnikova različica enaka 0 ali se je s spletno različico razlikovala za več kot 2000, je spletna aplikacija mobilni odgovorila z vsemi objekti, ki jih je uporabnik potreboval. V primeru, da se je uporabnikova različica s spletno

razlikovala za manj kot 2000 je spletna aplikacija odgovorila mobilni samo z objekti, ki jih mora ažurirati. Če pa sta kontrolni različici enaki, pa je spletna aplikacija odgovorila s statusom, da je podatkovna baza že ažurirana.

```
@api_view(['GET', 'POST'])
@permission_classes((IsAuthenticated,))
@login_required
def mobile_sync_db(request, company_id):

    seq = data['database_version']
    device_id = data['device_id']
    sync_objects = Sync.objects.only('seq')\
        .filter(company=company).order_by('-seq')
    last_key = 0
    if len(sync_objects) > 0:
        last_key = sync_objects[0].seq

    ret = {'version': last_key}

    if seq == 0 or last_key - 2000 > seq or seq > last_key:

        # if first sync or sync outdated, why not load whole DB

        ret['updated'] = False
        ret['drop'] = True

        # get categories
        categories = Category.objects.filter(company=company)
        ct = []

        for category in categories:
            ct.append(category_to_dict(category, android=True))

        ret['categories'] = ct

    elif seq < last_key:

        ret['updated'] = False
        ret['drop'] = False
        seq_list = Sync.objects.filter(company=company, seq__lte=last_key)

        items = []

        for seq_item in seq_list:
            item_ret = {'action': seq_item.action,
                        'model':  seq_item.model,
                        'object_id': seq_item.object_id}

            if seq_item.action == 'save':
                if seq_item.model == 'Category':
                    c = Category.objects.get(id=seq_item.object_id)
                    item_ret['item'] = category_to_dict(c, android=True)


        ret['items'] = items

    else:
        ret['updated'] = True

    return JsonOk(extra=ret, safe=False)
```

Slika 7.13: Del funkcije, ki vrača ažurirane podatkov.

# Poglavje 8

# Sklepne ugotovitve

V diplomskem delu sem predstavil tehnologije in postopke za implementacijo in medsebojno povezovanje mobilne in spletne aplikacije. Cilj diplomskega dela je bila implementacija spletnih storitev v okolju Django ter implementacija androidne aplikacije. Za medsebojno povezovanje teh dveh aplikacij sem uporabil pristop REST in ogrodje Django REST framework, ki sta skrbela za implementacijo APIjev, preko katerih mobilna aplikacija dostopa do strežnika. Tovrstni sistem omogoča, da lahko do istih podatkov, ki so shranjeni na strežniku dostopamo preko različnih naprav. Podrobneje sem predstavil implementacijo spletnih storitev, avtentikacijo uporabnika in ažuriranje podatkov.

Sama implementacija aplikacije ni delala večjih problemov, saj sem imel za implementacijo aplikacije v podjetju že predhodne izkušnje ter dovolj časa za izvedbo tega projekta. Poleg tega je na spletu veliko dokumentacije in spletnih forumov, ki sta mi olajševala delo.

Z največjim izzivom sem se soočil pri implementaciji sistema, ki je skrbel za ažurnost podatkov. Ker sem se s takim sistemom srečal prvič, sem moral sam najti rešitev, ki bi bila ustrezna. Idejo sem predstavil sodelavcem, kateri so mi jo z raznimi predlogi pomagali izboljšati. Sam sistem mi je predstavljal izziv tudi zaradi svoje obsežnosti.

Manjše težave mi je predstavljal tudi sistem za avtenticiranje, saj se zah-

teve za dostop preko Google prijave velikokrat spreminjajo. Ko so se zahteve spremenile, je storitev enostavno prenehala delovati. Poleg tega je prišlo v podjetju velikokrat do sprememb pri izvajanju funkcionalnosti na strežniku, brez da bi bile spremembe že v naprej predstavljene.

Realizacije takšnega sistema sem se lotil dobro premišljeno, vseeno pa bi v kakšnem primeru danes uporabil drugačen pristop. Prva stvar bi bila uporaba različnega okolja za razvoj androidne aplikacije, saj Eclipse uradno ni več podprt. Poleg tega sem se napačno lotil funkcionalnosti v androidni aplikaciji, ki so skrbele za povezovanje s spletno aplikacijo, saj so funkcije, ki sem jih uporabil opuščene. Funkcionalnosti bo zaradi tega potrebno napisati ponovno.

Končni sistem je zasnovan dobro, saj arhitekturna oblika REST skrbi, da lahko do istih funkcionalnosti dostopamo preko različnih platform. V spletni aplikaciji so implementirani vsi APIji, preko katerih dostopamo, ko želimo upravljati s podatki. Naš sistem bi izboljšal tako, da bi se lotil razvijanja aplikacij na drugih operacijskih sistemih, kot sta iOS in Windows Mobile. S tem bi podjetje pokrilo večji del naprav in in s tem pridobilo tudi večji tržni delež.

# Literatura


[1] [Online]. Dosegljivo: http://www.w3schools.com/webservices/ws_intro.asp [Dostopano 1.9.2015]

[2] Android (operating system). [Online]. Dosegljivo: http://en.wikipedia.org/wiki/Android_(operating_system). [Dostopano 11. 8. 2015].

[3] Android (operating system). [Online]. Dosegljivo: https://www.sqlite.org/about.html. [Dostopano 11. 8. 2015].

[4] Extensible Markup Language (XML). [Online]. Dosegljivo: http://www.w3.org/XML/. [Dostopano 13. 8. 2015].

[5] Soap vs REST. [Online]. Dosegljivo: http://www.javatpoint.com/soap-vs-rest-web-services. [Dostopano 31. 8. 2015].

[6] SOAP vs REST. [Online]. Dosegljivo: http://spf13.com/post/soap-vs-rest. [Dostopano 31. 8. 2015].

[7] [Online]. Dosegljivo: http://www.findalltogether.com/wp/webdevelopment/framework/laravel/django-vs-laravel-vs-rails/ [Dostopano 31.8.2015]

[8] [Online]. Dosegljivo: http://www.slant.co/topics/362/ [Dostopano 31.8.2015]

[9] [Online]. Dosegljivo:
http://www.w3schools.com/aspnet/mvc_intro.asp [Dostopano 25.8.2015]

[10] [Online]. Dosegljivo:
http://www.comentum.com/phonegap-vs-native-app-development.html [Dostopano 15.8.2015]

[11] [Online]. Dosegljivo:
http://www.xml.com/pub/a/ws/2001/04/04/soap.html [Dostopano 13.8.2015]

[12] [Online]. Dosegljivo:
http://www.w3schools.com/webservices/ws_intro.asp [Dostopano 13.8.2015]

[13] [Online]. Dosegljivo:
http://java.com/en/download/faq/whatis_java.xml [Dostopano 11.8.2015]

[14] [Online]. Eclipse (software) Dosegljivo:
https://en.wikipedia.org/wiki/Eclipse_(software) [Dostopano 11.8.2015]

[15] [Online]. Dosegljivo:
http://json.org/ [Dostopano 14.8.2015]

[16] Tools Help [Online]. Dosegljivo:
http://developer.android.com/tools/help/index.html [Dostopano 11.8.2015]

[17] Python (programming language) [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Python_(programming_language) [Dostopano 12.8.2015]

[18] PostgreSQL About [Online]. Dosegljivo: `http://www.postgresql.org/about/` [Dostopano 17.8.2015]

[19] Understanding SOAP and REST Basics And Differences (programming language) [Online]. Dosegljivo: `http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/` [Dostopano 13.8.2015]

[20] State of the developer nation Q1 2015. (2015). 1st ed. Vision mobile.